UNIVERSITÉ PARIS VI - PIERRE ET MARIE CURIE

U.F.R. D'INFORMATIQUE

THÈSE DE DOCTORAT

En vue de l'obtention du grade de
docteur de l'université Paris VI
en informatique

Présentée par

Pierre NGUYEN TUONG

DÉFINITION ET IMPLANTATION D'UN LANGAGE DE CONCEPTION DE COMPOSANTS ANALOGIQUES RÉUTILISABLES

Soutenue le 13 juin 2006 devant le jury composé de :

$M.$	Andreas KAISER	rapporteur
Mme	Noëlle LEWIS	rapporteur
Mme	Andreia CATHELIN	examinateur
$M.$	Patrick GARDA	examinateur
$M.$	Yves LEDUC	invité
$M.$	Alain GREINER	directeur de thèse
Mme	Marie-Minerve LOUËRAT	co-directrice de thèse

Résumé

La conception de circuits analogiques doit faire face à un triple problème : les plateformes de développement existantes traitent essentiellement la simulation électrique et le dessin des masques ; il existe peu d'algorithmes permettant d'automatiser le processus de création ; l'acceptation de nouveaux outils automatiques par la communauté de concepteurs va à l'encontre d'habitudes bien établies. Il en résulte que les connaissances acquises dans la conception d'un circuit ne sont pas transmises avec efficacité et sont souvent l'apanage de l'inventeur. Les réalités industrielles, où se manifeste la réutilisation dans de multiples contextes et pour des technologies différentes, impliquent une évolution méthodologique permettant un cycle de conception plus court grâce à la capitalisation des acquis.

Cette thèse propose une plateforme de conception analogique dont la méthodologie novatrice repose sur le concept de générateurs paramétrables et communicants. Le concepteur est amené à réaliser un circuit paramétrable, en termes de spécifications et de technologie, appelé générateur. L'exécution du générateur dans un contexte technologique et applicatif fournit une « netlist » dimensionnée et le dessin des masques. Pour concevoir un circuit paramétrable, cette thèse propose un langage adapté à la description des phases clefs de la conception que sont la description électrique (« netlist »), la description physique (« layout ») et l'exploration du domaine de conception. L'approche hiérarchique concourt à résoudre progressivement le problème du dimensionnement électrique, tout en autorisant une organisation claire des connaissances qu'il est alors possible de capitaliser. Une interaction forte entre le layout et la partie électrique permet de repousser la coûteuse phase d'extraction-simulation en fin de cycle de conception.

Mots clefs

Circuits intégrés analogiques CMOS, module de propriété intellectuelle, circuits analogiques hiérarchiques, expression et capitalisation de la connaissance, langage analogique, exploration du domaine de conception, co-conception physique-électrique, réutilisation.

Abstract

Analog integrated circuit design has to face three issues : existing specialized CAD plateform are mainly dedicated to electrical simulation and layout design ; there are few algorithms to ease design automation ; the acceptance of new EDA tools by the community of designers goes against well established practices. As a result, knowledge is kept inside a circuit design and is the prerogative of the inventor. It is against the trend which leads to massive intellectual property reuse required by process migration or specification migration.

This thesis proposes a new methodology involving parameterized communicating generators. The goal is to design a parameterized circuit, in terms of specification and technological process called "generator". Running a generator gives a sized netlist together with the layout. In order to design such a parameterized circuit, the three key stages of the design which are the electrical netlist description, the layout description and the design space exploration, are performed with a dedicated language described in this thesis. Using a hierarchical approach contributes to solve gradually and naturally the problem of electrical sizing while authorizing a clear organization of knowledge. Knowledge capitalization is thus allowed. The physical-electrical codesign makes it possible to push back the expensive phase of extraction-simulation at the very end of the design flow.

Keywords

Analog CMOS integrated circuit, intellectual property core, hierarchical analog circuit, knowledge capture, knowledge based design, Electronic Design Automation, analog language, design space exploration, physical-electrical codesign, reuse.

Simplifions, simplifions, simplifions...

À mes parents, à ma famille.

Remerciements

Je remercie très chaleureusement Alain Greiner et Marie-Minerve Louërat pour m'avoir accueilli au sein du laboratoire, et pour avoir fait en sorte que ma thèse se déroule dans d'excellentes conditions. J'apprécie tout particulièrement leurs qualités humaines sans égales, leurs capacités relationnelles sans limites et leurs grandes compétences.

Je suis reconnaissant à Andreas Kaiser, directeur de recherche CNRS au laboratoire IEMN-ISEN de Lille, Noëlle Lewis, maître de conférence au laboratoire IXL à Bordeaux, Andreia Cathelin, de STMicroelectronics, Patrick Garda, professeur UPMC au LISIF, Yves Leduc, de Texas Instruments, d'avoir bien voulu faire partie de mon jury et d'avoir partagé les mêmes centres d'intérêt, en particulier Noëlle Lewis et Andreas Kaiser pour leurs remarques constructives à propos de ce manuscrit.

Je pense à Marie-Martine, Roselyne, Patricia, Olivier, Hervé et Emmanuel, qui ont été mes collègues en enseignement et en recherche. Danièle et François, pour leur gentillesse et leur disponibilité. Frank, pour sa patience et sa bonne humeur. Nicole, Régine, Shahin, Renée, Willy et Elisabeth qui m'ont accompagné durant ces années. Pirouz, qui n'a pas réussi à me convertir à vi.

Le sport m'a accompagné : Maurice, mon premier Maître d'armes qui m'a fait découvrir l'escrime et qui m'a tant appris, Yoann et Xavier, frères d'arme au fleuret. Bien sûr, Soleiman, toujours disponible et grand aventurier. Alban, Sylvestre, Camille et tous les autres escrimeurs.

Je remercie Louis, Dominique, Alain, Jean-Louis, Jacques, Mickaël, Victor et Michel avec qui j'ai passé des moments qui m'auront marqué pour la vie à Lourcine, et aux côtés desquels je continue ma route.

Enfin et surtout, mille remerciements à mes parents, ma tante, ma famille qui, durant toutes ces années, m'ont soutenu, encouragé, réconforté et conseillé. Je leur doit d'être ici, c'est pour et grâce à eux que je continue. Je pense aussi aux miens, où qu'ils soient, en Australie, aux États-Unis, au Canada, au Philippines ou au Vietnam.

Table des matières

1 Introduction **1**

2 Problématique **7**
2.1 Motivations 8
2.2 Objets manipulés 10
2.2.1 Aspect électrique 11
2.2.2 Aspect physique 11
2.3 Flot classique de conception analogique 12
2.4 Conclusion 14

3 État de l'Art **15**
3.1 Réalisations industrielles 16
3.1.1 Virtuoso®Neocell/Virtuoso Neocircuit de Cadence® 16
3.1.2 Barcelona Design 18
3.1.3 Sagantec 19
3.2 Réalisations universitaires 19
3.2.1 OASYS 19
3.2.2 OPASYN et ANACONDA 19
3.2.3 KOAN-ANAGRAM II 20
3.2.4 AMGIE 20
3.2.5 BALLISTIC 20
3.2.6 PAD 22
3.2.7 Océane 22
3.2.8 Microwind 23
3.3 Résumé 23

4 Principes généraux du langage CAIRO+ **25**
4.1 Méthode de conception proposée 26
4.2 Le précurseur : le langage CAIRO 29
4.3 Schéma non dimensionné 29
4.4 Placement relatif 30
4.5 Dimensionnement électrique 32
4.6 Génération du dessin des masques 33

4.7 Prise en compte des capacités parasites . . . 34
4.8 Migration technologique . . . 35
4.9 Conclusion . . . 37

5 Exploration du domaine de conception 39
5.1 Présentation générale . . . 40
5.2 Étapes du DSES . . . 41
5.3 Interface fonctionnelle . . . 42
5.3.1 Procédures . . . 43
5.3.2 Paramètres . . . 44
5.4 Exploration du domaine de conception : DSES . . . 46
5.4.1 Communication descendante . . . 46
5.4.2 Communication ascendante . . . 47
5.4.3 Mise en œuvre de l'exploration du domaine de conception . . . 48
5.5 Résumé . . . 50

6 Placement optimisé d'objets déformables 51
6.1 Analyse du problème . . . 52
6.2 Fonction de forme d'un dispositif élémentaire . . . 53
6.3 Fonctions de Forme sous contrainte FFDX et FFDY . . . 54
6.4 Construction ascendante de la Fonction de Forme d'un module . . . 55
6.4.1 Fonctions de forme sous contrainte d'un conteneur vertical . . . 55
6.4.2 Fonctions de forme sous contrainte d'un conteneur horizontal . . . 56
6.4.3 Fonction de forme d'un module . . . 56
6.5 Propagation descendante des contraintes de forme . . . 57
6.6 Implantation . . . 57
6.6.1 Structure de données . . . 57
6.6.2 Algorithme de composition pour un conteneur horizontal . . . 59
6.6.3 Exemple . . . 60
6.7 Conclusion . . . 61

7 Prise en compte des capacités parasites dans la phase de dimensionnement électrique 63
7.1 Capacités parasites . . . 64
7.1.1 Routage . . . 64
7.1.2 Dispositifs élémentaires . . . 65
7.2 Interaction des vues physiques et électriques . . . 66
7.2.1 Calcul des capacités parasites . . . 66
7.2.2 Prise en compte des capacités parasites . . . 68
7.3 Conclusion . . . 69

8 CAIRO+ : une plateforme de développement 71
8.1 Concepts généraux . . . 72
8.1.1 Programme exécutable . . . 73
8.1.2 Phases du flot de conception et fonctions associées . . . 73
8.2 Création des structures de données . . . 74
8.2.1 Déclaration des composants . . . 74
8.2.2 Patron électrique . . . 75
8.2.3 Patron physique . . . 76
8.2.4 Interface fonctionnelle . . . 79
8.3 Exploration du domaine de conception : DSES . . . 81
8.3.1 Corps d'une procédure . . . 81
8.3.2 Communication descendante : passage des arguments . . . 81
8.3.3 Communication ascendante : transmission des résultats . . . 83
8.3.4 Gestion des échecs . . . 86
8.4 Formation d'un exécutable . . . 90
8.5 Résumé . . . 94

9 Exemples et résultats 95
9.1 Conception hiérarchique d'un OTA . . . 96
9.1.1 Présentation d'un OTA . . . 96
9.1.2 Choix des dispositifs élémentaires . . . 98
9.1.3 Équations de l'OTA . . . 100
9.1.4 Dimensionnement électrique de l'OTA . . . 101
9.1.5 Procédure $VP1$ . . . 103
9.1.6 Présentation du code réalisant l'OTA . . . 107
9.1.7 Simulations et résultats . . . 110
9.2 Conception hiérarchique d'un filtre passe-bas pour un modulateur $\Sigma\Delta$ à temps continu et mode courant . . . 119
9.2.1 Objectif et architecture . . . 119
9.2.2 Hiérarchie d'un module « Filtre passe-bas » . . . 121
9.2.3 Dimensionnement électrique . . . 122
9.2.4 Simulations et résultats . . . 126
9.3 Conclusion . . . 135

10 Conclusion 139

A Plateforme de tests l'OTA 145
A.1 Mesure de la tension de décalage . . . 145
A.2 Analyse petit signal . . . 147

B Mode d'emploi de libMOS 149

C Mode d'emploi de l'OTA 161

D Code source du générateur OTA **167**

E Code source du fichier de tests de l'OTA en $0,13\ \mu m$ **199**

F Code source du générateur *gmc* **203**

G Code source du fichier de tests du *gmc* en $0,13\ \mu m$ **219**

H Code source du générateur filtre **225**

I Code source du fichier de tests du filtre en $0,13\ \mu m$ **235**

J Publications **241**

Bibliographie **243**

Table des figures

2.1 Le meilleur des deux mondes . . . 8
2.2 Boucle système-circuit-layout . . . 9
2.3 Générateur typique utilisé avec deux spécifications et deux procédés cibles 10
2.4 Un transistor NMOS et ses régimes de fonctionnement avec $V_{BS} = 0$ . . . 11
2.5 Technique de repliement . . . 12
2.6 Paire différentielle repliée . . . 12
2.7 Flot de conception classique . . . 13

3.1 Flot de conception Virtuoso Neocircuit [Cad] . . . 17
3.2 Flot de conception Virtuoso Neocell [Cad] . . . 17
3.3 Flot de conception Barcelona Design [Des] . . . 18
3.4 Amplificateur opérationnel dans la même technologie pour deux hauteurs de cellules différentes, dessin des masques réalisé par Ballistic [ODJ+95] [OM95] . . . 21

4.1 Décomposition en dispositifs élémentaires et icône associée à un amplificateur opérationnel cascode replié à transconductance (OTA) selon [IF90] (page 107) . . . 27
4.2 Étapes de conception dans le flot CAIRO+ . . . 28
4.3 Schéma électrique d'un OTA de type P . . . 30
4.4 Arbre des modules pour un OTA Miller . . . 30
4.5 Arbre des conteneurs pour un OTA Miller . . . 31
4.6 Communication entre générateurs . . . 33
4.7 Deux réalisations d'un même routage procédural . . . 35
4.8 Abstraction du modèle électrique . . . 36
4.9 Illustration de la migration technologique . . . 37

5.1 Domaine de conception . . . 40
5.2 Parcours de l'arbre des modules lors d'un DSES . . . 42
5.3 L'interface fonctionnelle . . . 43
5.4 Principe de fonctionnement d'une procédure . . . 44
5.5 Communication descendante . . . 47
5.6 Communication ascendante . . . 48
5.7 Organigramme d'un DSES et les blocs de contrôle . . . 49

6.1 Illustration du repliement d'un transistor 53
6.2 Fonction de forme 54
6.3 Fonctions de forme DXFFX et DYFFX sous la contrainte DX 54
6.4 Fonctions de forme sous la contrainte DY 55
6.5 Représentation en mémoire d'une fonction de forme 58
6.6 Composition de deux fonctions de forme 61

7.1 Exemple de routage procédural 65
7.2 Fil à quatre segments 65
7.3 Quelques capacités parasites pour un transistor NMOS 66
7.4 Rétro-annotation des capacités parasites de routage et utilisation 67
7.5 Calcul des capacités parasites pour un dispositif élémentaire 68
7.6 Boucle pour la prise en compte des capacités parasites 69

8.1 Étapes de conception et fonctions associées 72
8.2 Un exécutable 73
8.3 Patron électrique de l'OTA de type P 77
8.4 L'environnement du conteneur 77
8.5 Patron physique de l'OTA de type P 78
8.6 Communication descendante pour un OTA 82
8.7 Communication ascendante pour un OTA 84
8.8 Le placement et le routage sont déclenchés par la fonction CAIRO_RESHAPE 91

9.1 Amplificateur différentiel à transconductance 96
9.2 OTA à transconductance classique 97
9.3 Schéma de principe de l'amplification 97
9.4 Transistor MOS et son modèle simplifié 98
9.5 OTA de type P - Tensions et courants associés 99
9.6 OTA de type N - Tensions et courants associés 99
9.7 Les deux étapes d'analyse d'un amplificateur 101
9.8 Procédures UGF 106
9.9 Évolutions du gain (à gauche) et de la phase (à droite) en fonction de la fréquence en 0.13 μm 115
9.10 Dessin des masques des deux OTA en $0,6$ μm et $0,13$ μm (échelle relative respectée) 116
9.11 Fréquence de transition et phase pour le schéma électrique dimensionné extrait avec capacités parasites de routage pour les deux spécifications en $0,13$ μm 118
9.12 OTA pour deux facteurs de forme et deux modes de réalisation 119
9.13 Densité spectrale typique d'un modulateur $\Sigma\Delta$ 120
9.14 Architecture d'un modulateur $\Sigma\Delta$ à temps continu d'ordre 3 120
9.15 Topologie d'un intégrateur de courant d'ordre 1 121
9.16 Hiérarchie d'un générateur d'intégrateur d'ordre 1 121
9.17 Procédures de dimensionnement du filtre et des intégrateurs 127

9.18 Bruit thermique et bruit en $\frac{1}{F}$ 128
9.19 Erreur relative sur UGF pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec OSR=64 131
9.20 Évolution du courant et de la capacité d'intégration pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec OSR=64 131
9.21 Écart en db entre $SNR_{Th+\frac{1}{F}}$ et $SNR_{Q_{mes}}$ et le SNR alloué pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec OSR=64 132
9.22 Mesure de la densité spectrale pour RESOLUTION=11 bits et BW=100 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 . . . 135
9.23 Dessin des masques du filtre pour RESOLUTION=11 bits et BW=100 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 136
9.24 Mesure de la densité spectrale pour RESOLUTION=8 bits et BW=50 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 136
9.25 Dessin des masques du filtre pour RESOLUTION=8 bits et BW=50 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 137

A.1 OTA de type P — Mesure de la tension de décalage 146
A.2 OTA de type P — Analyse petit signal 146

Liste des tableaux

3.1 Comparaison des possibilités des outils présentés 24

5.1 Type d'un paramètre 45

7.1 Stabilisation des paramètres en technologie 0,6 μm 70

9.1 Procédures de l'OTA 102
9.2 Valeur des paramètres de l'OTA 112
9.3 Performances comparées de l'OTA : UGF (MHz), PM (degré) et GAIN (db) 112
9.4 Comparaison des paramètres petits signaux de l'OTA 114
9.5 Écart dans les tailles calculées et effectivement générées des transistors . . 115
9.6 Valeur des paramètres de l'OTA 117
9.7 Performances comparées de l'OTA : UGF (MHz), PM (degré) et GAIN (db) 117
9.8 Écart dans les tailles calculées et effectivement générées des transistors . . 118
9.9 Paramètres du filtre passe-bas 122
9.10 Contraintes sur les paramètres 123
9.11 Paramètres de l'intégrateur 124
9.12 Valeurs des paramètres d'entrée du premier gmc 128
9.13 Évolution de la fréquence de transition UGF, de la capacité d'intégration CAPINT et du courant IBIAS en fonction de la bande passante pour divers rapports signal/bruit pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 130
9.14 Rapports SNR en db résultant de la précision demandée ($Budget$), du bruit de circuit calculé ($SNR_{Th+\frac{1}{F}}$) et du bruit de quantification mesuré (Q_{mes}) pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64 133
9.15 Valeur des paramètres d'entrée du filtre 134

Chapitre 1

Introduction

Le monde de la micro-électronique évolue rapidement. Le temps où les masques d'un circuit étaient tracés à la main a fait place à la conception assistée par ordinateur ; malheureusement, le gain de productivité acquis ne compense pas le changement constant de technologie, qui évolue plus vite que les outils destinés à l'utiliser. Dans ce domaine, comme dans bien d'autres de l'informatique, la capitalisation des acquis est devenue une nécessité. Il s'agit de réduire le temps de mise sur le marché (*time to market*) en réutilisant un composant afin de pouvoir construire des systèmes de plus en plus complexes se basant sur des éléments éprouvés et parfaitement caractérisés. La part de l'analogique en terme de temps et de difficulté de conception, largement supérieure à celle du numérique pour un nombre inférieur de transistors, rend la problématique encore plus aiguë.

Le domaine de la conception numérique s'est longtemps opposé au domaine de la conception analogique par ses méthodes, les outils employés et les acteurs. Les méthodes et les outils sont intimement liés : de nombreux algorithmes ont été développés pour la résolution des problèmes liés à la conception numérique, les outils existent et sont performants. L'expérience du concepteur, nécessaire, n'est pas aussi critique que dans la conception analogique où peu de méthodes ont été formalisées et où, en conséquence, les outils automatiques sont peu présents.

Cet état de fait réside dans la spécificité de la conception analogique, ou plus exactement dans la nature continue des informations à traiter, par opposition à un système discret comme peut l'être un circuit numérique. L'élément de base du circuit numérique est souvent — au niveau masques physiques comme au niveau synthèse — une cellule précaractérisée faisant partie d´une bibliothèque, ou même un générateur pour les structures régulières telles que les RAM ou les ROM. Au niveau synthèse, il s'agit d'une fonction logique plus ou moins complexe. Au niveau physique, la cellule est constituée par le dessin des masques. D'un contexte d'utilisation à un autre, d'un procédé technologique à un autre, la fonction synthétisée est identique tandis que le dessin des masques peut à nouveau être généré par un outil automatique. Il suffit que les transistors possèdent un domaine électrique de fonctionnement suffisant pour assurer la réalisation de la fonction logique (passant, bloqué). En ce sens, la portabilité est assurée par les outils de synthèse qui utilisent des générateurs ou des bibliothèques de cellules précaractérisées adaptés à la technologie cible.

Les marges définissant l'état d'un transistor ne peuvent plus être invoquées dans le cas d'un circuit analogique où les signaux sont continus — ce qui implique une variation continue des spécifications des objets utilisés —, et où ces transistors doivent être parfaitement dimensionnés pour fournir un courant précis. Il existe bien des cellules précaractérisées, mais elles ne sont pas portables, car leurs caractéristiques sont figées. Une simple ho-

mothétie du dessin des masques n'est pas envisageable, c'est le système entier qu'il faut dimensionner afin qu'il satisfasse des spécifications précises. Or ce dimensionnement fait partie d'un long processus de conception : définition des contraintes, choix de la topologie du circuit, dimensionnement électrique des éléments, placement, routage, extraction des éléments parasites, simulation électrique. Si les spécifications ne sont pas atteintes, le dimensionnement électrique est de nouveau réalisé en tenant compte des parasites. Sachant qu'une simulation électrique peut prendre plusieurs jours (lorsque elle est faisable, c'est à dire pour un nombre restreint de transistors dans le cas d'une simulation transitoire), que la conception d'un circuit repose essentiellement sur l'expérience de l'analogicien (pas de méthodes générales), on comprendra aisément les difficultés rencontrées. À cela s'ajoute l'évolution des technologies vers une finesse de gravure de plus en plus petite : les approximations faites en numérique quant aux capacités de couplage, aux résistances des fils... ne sont plus possibles — ces effets sont devenus prépondérants alors qu'ils étaient auparavant approchés, voir négligés. L'analogique y est encore plus sensible et souffre encore moins les approximations, surtout lorsque les contraintes sont la fréquence et la consommation. Même si l'on cherche à réaliser un maximum de fonctions en numérique, on ne peut se passer de l'interface avec un « monde analogique », le développement des communications ne faisant qu'accroître ce besoin. De nombreux produits de grande consommation connaissent une utilisation nomade et doivent dialoguer entre eux ou avec un système sédentaire ; la simple utilisation d'un téléphone mobile met en jeux une chaîne complexe où les informations sont véhiculées par différents média. L'analogique, présente dans toutes les interfaces, devient un facteur limitant dans le flot de conception d'un circuit mixte numérique-analogique en terme de temps de développement, de surface et de difficulté de réalisation.

*

L'objectif est de réaliser l'équivalent analogique du couple cellules précaractérisées/générateurs numériques afin de pouvoir mettre en œuvre une politique de réutilisation. Le caractère dynamique exposé précédemment pousse naturellement vers la conception de générateurs (modules de propriété intellectuelle analogiques — *Analog Intellectual Property Cores*) dont on peut d'ores et déjà esquisser les caractéristiques :

- La description doit se faire à l'aide d'un langage simple. On choisira de préférence un langage préexistant pour lequel un compilateur performant a été développé.
- Le générateur favorisera la conception hiérarchique et devra permettre la migration technologique.
- Le générateur doit offrir la possibilité de capitaliser l'expérience.
- Le générateur doit pouvoir permettre l'exploration du domaine de conception de manière automatique en faisant interagir les différents niveaux hiérarchiques.
- Le langage doit permettre la mise à jour et la modification.

Cette thèse propose la définition d'un langage de conception analogique répondant aux critères énoncés. Il y est démontré qu'il est possible de créer des modules réutilisables non seulement dans différentes technologies, mais aussi dans différents contextes de spécification et ce sans autre effort que celui de l'écriture initiale.

*

Si l'on exclut le VHDL-AMS et le VERILOG-AMS, essentiellement orientés vers la modélisation de plus ou moins haut niveau, il existe plusieurs langages de description matérielle tournés vers la conception de générateurs analogiques. Outre ceux inclus dans les outils commerciaux (Skill de Cadence[Inc01]), un petit nombre se compte dans le monde universitaire (Ballistic[Mar]). La plupart ne prend pas en compte la phase de synthèse électrique mais s'attache au dessin des masques. Dans cette catégorie, le laboratoire LIP6/ASIM a aussi développé en 1995 son propre langage de génération automatique, CAIRO [1][Des01]. Cette première tentative a permis la réalisation d'un modulateur sigma-delta à faible tension d'alimentation dont la partie synthèse fut prise en charge par des procédures proposées par Mohamed Dessouky, procédures faisant appel à des fonctions de Comdiac (ENST), maintenant Océane[Por05]. La partie dessin des masques fut assurée par CAIRO, tandis que le circuit fut finalisé sous Cadence.

Ce processus de conception, qui se rapproche du flot traditionnel de conception, n'était pas véritablement automatisé. On peut aussi ajouter que la réalisation physique du circuit influençant grandement les performances (symétries, capacités parasites...), il est devenu essentiel de coupler la phase de synthèse électrique avec la génération du dessin des masques. CAIRO+ est le résultat de cette évolution.

Le travail décrit dans cette thèse s'est articulé autour de cinq points :

- étudier et formaliser le flot de conception analogique de manière à l'automatiser ;
- étudier les règles régissant la migration technologique et la migration de spécifications ;
- définir les interactions entre le dessin des masques et la synthèse électrique ;
- dégager l'architecture d'un générateur ;
- définir et concevoir une plateforme s'appuyant sur un langage permettant d'implanter des générateurs, tester et valider les solutions retenues.

Une bibliothèque de dispositifs élémentaires (éléments terminaux) et de modules a été développée grâce à l'utilisation du langage CAIRO+ créé pour l'occasion.

*

[1] Creating Analog Ips for Reuse with Optimisation

Le plan du manuscrit s'articule autour de dix chapitres :

- Le chapitre 1 est le présent chapitre.
- Le chapitre 2 pose la problématique et expose les motivations de ce travail. Les modules réutilisables sont nécessaires, car les aspects liés aux objets manipulés et les méthodes de conception diffèrent du monde numérique. Les entités élémentaires sont déformables tandis qu'il faut assurer le lien entre le niveau système, le circuit et le dessin des masques. Il faut aussi prévoir la migration technologique et le changement de spécification. Face à une méthodologie peu automatisée/automatisable, le processus classique de conception doit être revu.
- Le chapitre 3 présente les réalisations commerciales et universitaires. Parmi les réalisations commerciales, Barcelona Design[Des] et Neolinear[Cad] sont avancés, mais les technologies utilisées sont propriétaires. Ces outils ne permettent pas à l'utilisateur de créer ses propres générateurs (ils sont fournis), ce qui entraîne un manque de contrôle. Les outils universitaires, dont Koan/Anagram[CGRC91], sont de deux types : soit basés sur un simulateur, soit sur une formulation analytique du problème.
- Le chapitre 4 dégage les conditions nécessaires à la réalisation d'un générateur portable. Il faut permettre au concepteur d'exprimer son expertise d'une manière qui lui permette d'automatiser facilement l'exploration du domaine de conception. Le concepteur de générateurs doit pouvoir intervenir à tous les niveaux. Bien sûr, le générateur réalisé doit assurer la portabilité, la réutilisation dans différents contextes et la migration technologique.
- Le chapitre 5 présente l'exploration du domaine de conception et les concepts afférents : les services rendus par le générateur, les paramètres attachés aux procédures de dimensionnement, la communication hiérarchique et la manière d'employer et transmettre des paramètres de haut niveau (niveau système) aux niveaux inférieurs jusqu'à arriver aux paramètres purement physiques. Les conditions d'indépendances vis à vis de la technologie et du contexte d'utilisation sont exposées.
- Le chapitre 6 présente la phase de placement. Les dispositifs élémentaires utilisés autorisant plusieurs formes possibles, il est nécessaire de gérer les différents facteurs de forme pour aboutir au placement du circuit complet. La notion de fonction de forme introduite dans ce chapitre permet de prendre en charge le placement en utilisant une propagation hiérarchique de la contrainte.
- Le chapitre 7 expose les interactions entre le dimensionnement électrique et la réalisation physique. La plupart des outils traite le dimensionnement électrique indépendamment de la réalisation physique bien que les deux soient fortement liés et s'influent mutuellement. CAIRO+ apporte la prise en charge simultanée de ces éléments par construction grâce à la méthodologie mise en place.
- Le chapitre 8 traite de l'implantation dans CAIRO+ de tous les principes qui ont été dégagés. Un générateur se compose de trois fonctions qui sont le reflet des étapes de dimensionnement : une fonction de création où les structures de données sont créées (instanciations, liste des signaux, placement relatif, déclaration des paramètres de

synthèse), une fonction d'exploration du domaine de conception, qui constitue le coeur du dimensionnement électrique et une fonction de routage.

- Le chapitre 9 présente les résultats sur des exemples réels. La conception d'un module de premier niveau basé sur un OTA est décrite en détail. Des modules plus complexes tels qu'un intégrateur « GM/C » en mode courant et un modulateur sigma-delta sont aussi commentés.
- Le chapitre 10 forme la conclusion de ce manuscrit et dévoile l'avenir de CAIRO+.

CAIRO+ doit compter sur l'adhésion de ses principaux utilisateurs, les concepteurs analogiciens. La question de savoir si CAIRO pouvait être utilisé par une personne autre que son concepteur avait été posée lors de la soutenance de thèse de Mohamed Dessouky. La présente thèse a pour but d'y apporter une réponse que l'on espère être « Tous les analogiciens peuvent utiliser CAIRO+ »

Chapitre 2

Problématique

Sommaire

2.1 Motivations 8
2.2 Objets manipulés 10
2.2.1 Aspect électrique 11
2.2.2 Aspect physique 11
2.3 Flot classique de conception analogique 12
2.4 Conclusion 14

La réutilisation est un concept clef du génie logiciel, et elle le devient de plus en plus en micro-électronique. Il existe peu d'algorithmes formalisant la conception automatisée d'un circuit analogique. Ce chapitre présente le flot de conception traditionnel d'un circuit analogique et en dégage les faiblesses.

2.1 Motivations

Les systèmes intégrés sur puce font de plus en plus appel à des systèmes mixtes analogiques/numériques. La tendance à l'intégration des composants analogiques passifs et actifs est de plus en plus forte, y compris jusqu'aux antennes dans le domaine des hyperfréquences. La partie numérique utilise des composants réutilisables de type *IP Core* (*Intellectual Property Core*) à tous les niveaux : bibliothèques de cellules précaractérisées, systèmes entiers synthétisables. Le temps de conception en est d'autant réduit, les erreurs sont minimisées. Le monde analogique a besoin de tels composants mais doit faire face au manque d'outils de synthèse automatique et aux méthodes spécifiques de dimensionnement électrique qui dépendent fortement de l'expertise du concepteur (figure 2.1 [1]). Tous les fondeurs proposent des fonctions analogiques particulières (puces *bluetooth*, *tuner* satellite...), mais la migration technologique est extrêmement coûteuse car ces composants doivent être entièrement redessinés et caractérisés pour chaque nouvelle génération du procédé de fabrication. Que ce soient les circuits dont les performances ne sont pas trop critiques (*commodity IP* — DAC, ADC) ou les circuits critiques (*star IP*), le problème de la réutilisation des composants analogiques est un problème non résolu. Plusieurs facettes du flot de conception analogiques sont alors à considérer.

FIG. 2.1: Le meilleur des deux mondes

Les aspects « système », « circuit » et « layout » (dessin des masques) sont intimement liés. Le choix des blocs fonctionnels entraîne des répercussions sur le circuit, qui impose des contraintes physiques de placement et de routage. À son tour, le dessin des masques peut entraîner une modification du « système » afin de satisfaire les contraintes électriques

[1] À gauche, tour Jussieu, Jussieu, Paris, France (Édouard Albert, architecte) ; à droite, tour Dubuffet, Issy les Moulineaux, France (réalisation Jean Dubuffet). Photographies ©Pierre Nguyen Tuong, 2005.

imposées. Une boucle de conception (exploration du domaine de conception) émerge donc de ce schéma, boucle d'autant plus difficile à satisfaire qu'elle est en partie dépendante du savoir-faire du concepteur analogicien (figure 2.2). Il existe une nécessité d'automatiser le processus.

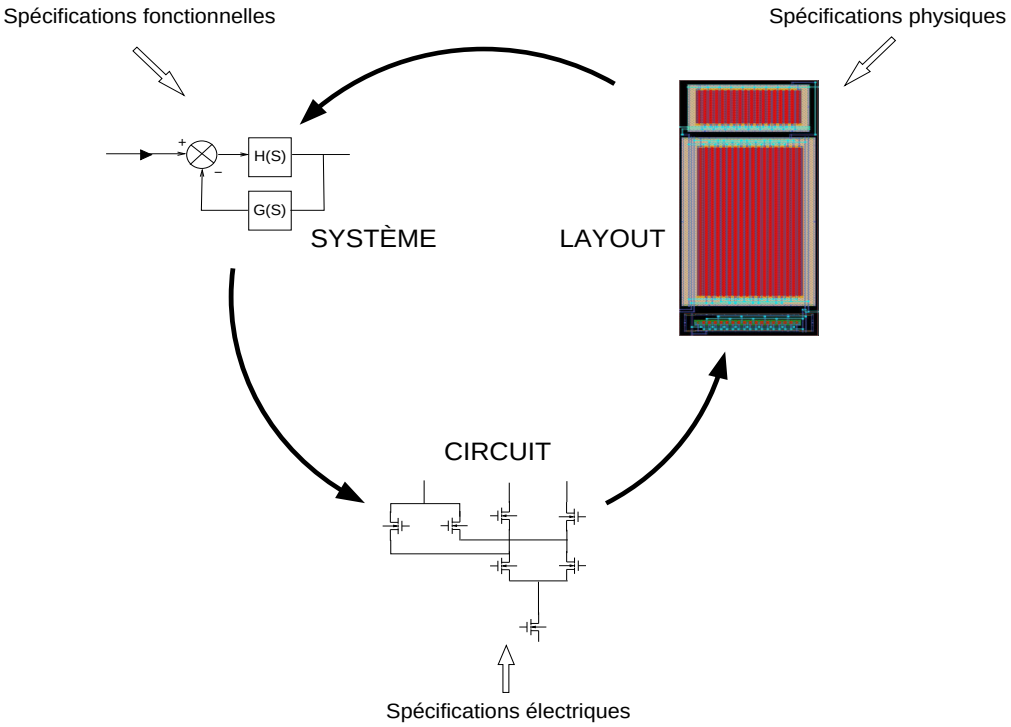


FIG. 2.2: Boucle système-circuit-layout

Le défi consiste à fournir au concepteur un langage lui permettant de formaliser sa propre expertise afin de réaliser l'exploration automatique du domaine de conception d'une manière qui assure la réutilisation et la portabilité du double point de vue des spécifications fonctionnelles d'une part et de la migration technologique d'autre part. La figure 2.3 présente une vue simplifiée d'un générateur et de ses contraintes associées. Le moteur qui déroule les phases de conception (création des structures de données, dimensionnement électrique, choix du dessin des masques suivant la contrainte de forme, routage et rétro-annotation des parasites) représente le cœur du générateur. Les spécifications fonctionnelles et la technologie cible (paramètres technologiques, règles de dessin réelles et facteur de forme) sont les éléments variables dont il faut tenir compte. Par spécification fonctionnelle, on entend les performances attendues et les contraintes appliquées aux fonctions analogiques ; par exemple, obtenir un filtre passe-bas avec une fréquence de coupure déterminée et une consommation maximale garantie. Les paramètres technologiques comprennent les valeurs physiques des paramètres dont dépendent les équations régissant le comportement électrique des composants élémentaires : un modèle de transistor comme BSIM4 est tributaire de plus de 300 paramètres fournis par le fondeur.

La procédure de dimensionnement électrique est le résultat de l'application des connais-

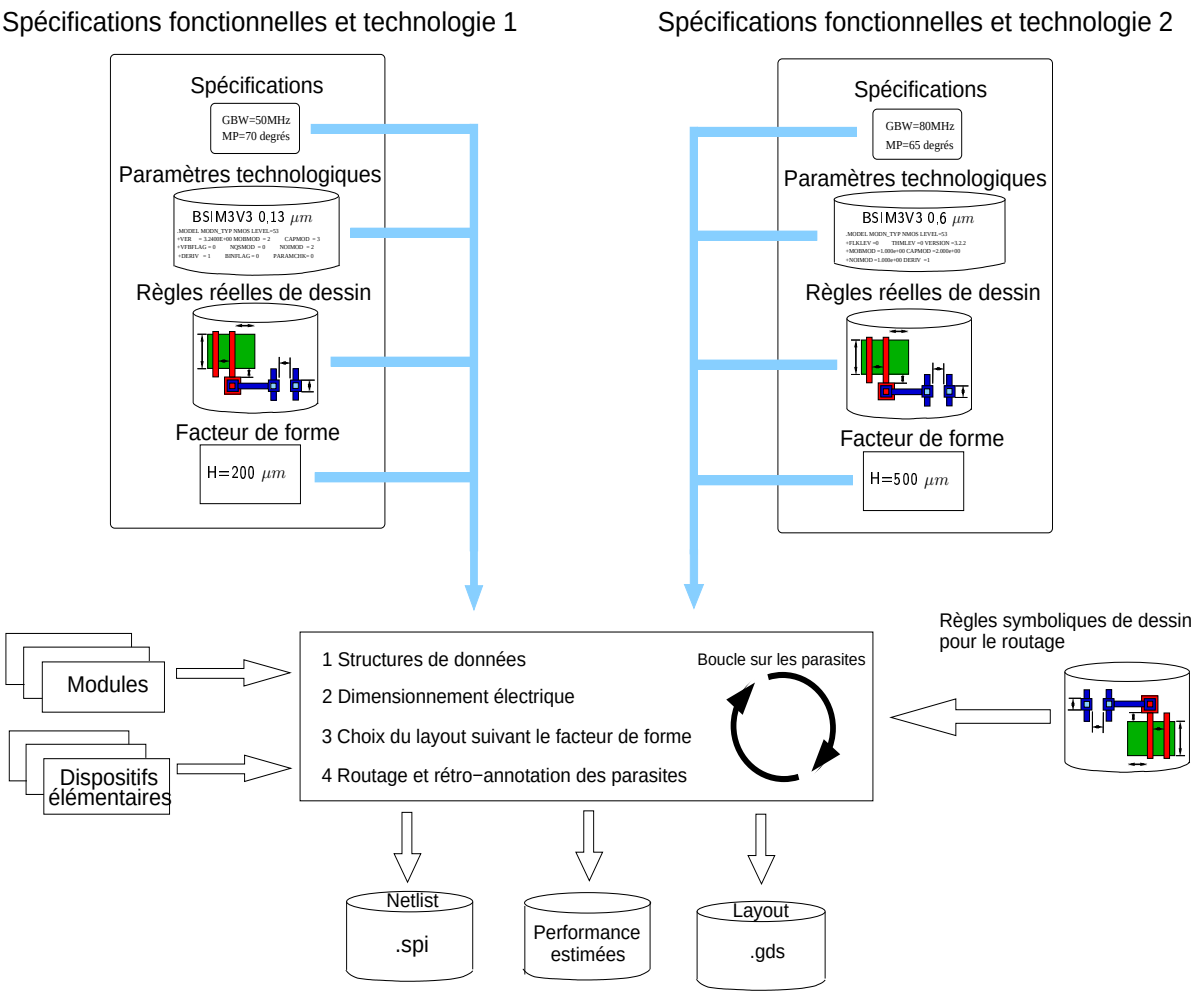


FIG. 2.3: Générateur typique utilisé avec deux spécifications et deux procédés cibles

sances du concepteur analogicien ; elle n'est pas la conséquence d'un algorithme de synthèse général comme il en existe en numérique. Cette situation rend délicate l'automatisation du processus de conception. Néanmoins, il est possible de contourner cet écueil par une approche de type conception de générateurs à condition de savoir exprimer et conserver l'expertise du concepteur ou de l'aider à formaliser ses méthodes de travail.

2.2 Objets manipulés

Les spécifications électriques des circuits numériques n'ont pas les mêmes objectifs que celles des circuits analogiques. Le monde numérique est un monde discret où les signaux ne peuvent prendre qu'un nombre fini de valeurs discrètes.

2.2.1 Aspect électrique

Le transistor NMOS représenté à la figure 2.4 possède un réseau de caractéristiques

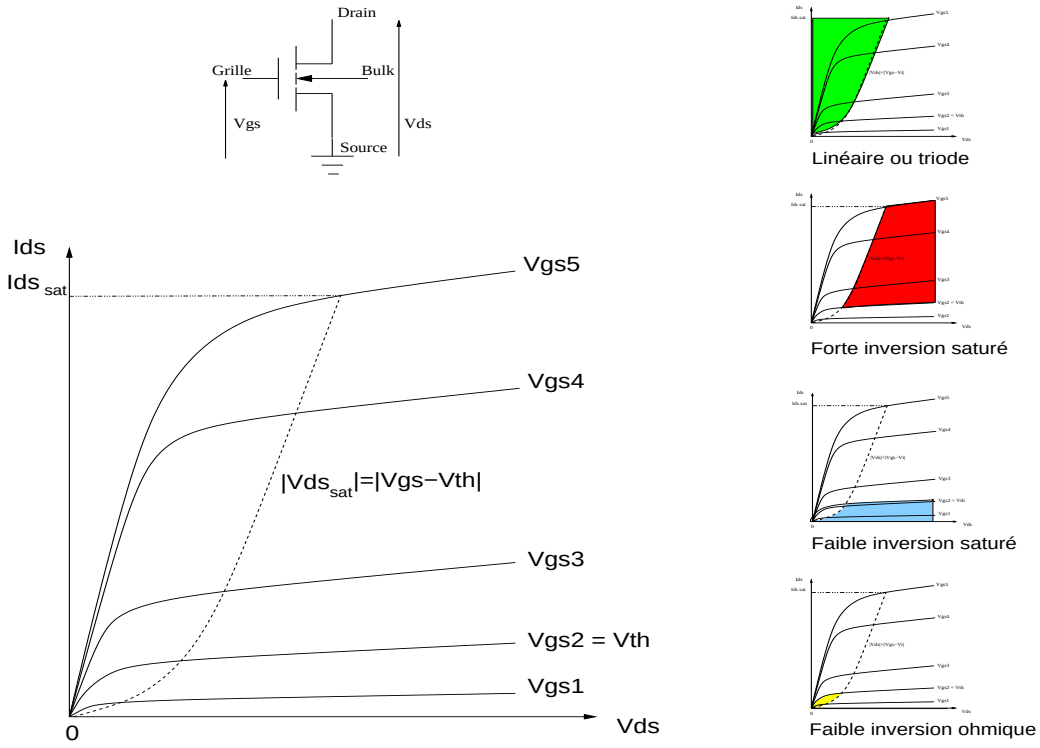


FIG. 2.4: Un transistor NMOS et ses régimes de fonctionnement avec $V_{BS} = 0$

qui exprime le lien entre la tension V_{DS} appliquée au drain et le courant de drain I_{DS} en fonction de la tension de grille V_{GS}. Pour une tension V_{GS} donnée il existe quatre régimes de fonctionnement :

- le régime linéaire lorsque $0 < V_{DS} < V_{DS_{sat}}$ et $V_{GS} > V_{TH}$;
- le régime de forte inversion saturé lorsque $V_{DS} > V_{DS_{sat}}$ et $V_{GS} \geq V_{TH}$;
- le régime de faible inversion saturé lorsque $V_{DS} > V_{DS_{sat}}$ et $V_{GS} < V_{TH}$;
- le régime de faible inversion ohmique lorsque $V_{DS} < V_{DS_{sat}}$ et $V_{GS} < V_{TH}$.

avec $V_{DS_{sat}} = V_{GS} - V_{TH}$, V_{TH} étant la tension de seuil. Considérant une technologie particulière, si l'on choisit un V_{GS} et une tension de polarisation V_{DS} fixes permettant un régime de fonctionnement se situant dans la zone saturée, le transistor NMOS se comporte comme un interrupteur pour un numéricien, et pour le concepteur analogicien comme un amplificateur de transconductance.

2.2.2 Aspect physique

Le facteur de forme (rapport hauteur/largeur) des composants analogiques n'est pas fixe pour une spécification électrique donnée : en appliquant des techniques de replie-

ment, on peut modifier la forme d'un transistor sans modifier ses caractéristiques électriques principales, même si les éléments parasites (capacités entre autres) varient. Pour un transistor de largeur W_T, il s'agit de mettre en parallèle un nombre N de transistors élémentaires dont la largeur unitaire W_N est égale à W_T/N. Les drains sont partagés, diminuant ainsi la valeur des capacités parasites par rapport au substrat. La figure 2.5 illustre cette technique, tandis que la figure 2.6 présente une paire différentielle repliée pour deux facteurs de forme différents dans la même technologie (0,13 μm).

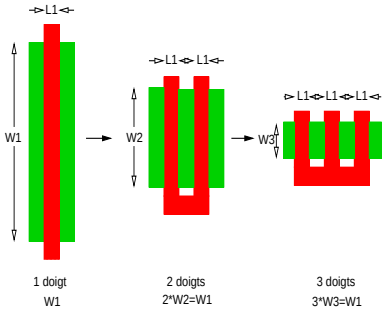


FIG. 2.5: Technique de repliement

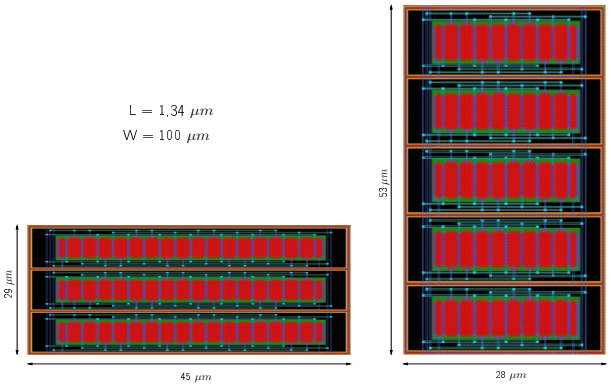


FIG. 2.6: Paire différentielle repliée

Il est donc possible de jouer sur le repliement pour obtenir un facteur de forme particulier.

2.3 Flot classique de conception analogique

Le concepteur analogicien suit une série d'étapes qui ne sont généralement pas automatisées (c'est à dire qu'il n'existe pas d'outils automatiques pour les prendre en charge ou que ceux-ci — s'ils existent — ne font pas partie d'un ensemble homogène) : la conception d'un circuit commence par le choix du système à concevoir, sa fonction et les objectifs à respecter (figure 2.7). L'analogicien détermine un schéma de principe du circuit qu'il décrit en général en *spice*[Nag75] afin de pouvoir effectuer des simulations électriques. Le dimensionnement électrique est une phase critique. L'objectif est de satisfaire l'ensemble des spécifications électriques tout en tenant compte des effets parasites induits par les imperfections du dessin des masques (capacités parasites). La méthodologie classique utilise donc une boucle où intervient une coûteuse phase de simulation qui se base sur les informations extraites du dessin des masques généré. Différents outils interviennent dans ce flot :

- La génération du dessin des masques est actuellement en partie automatisée : un circuit analogique est le plus souvent conçu à l'aide d'un éditeur de dessin des masques pour placer et router les composants de base — *PCells* de Cadence par exemple — (transistors, résistances...).

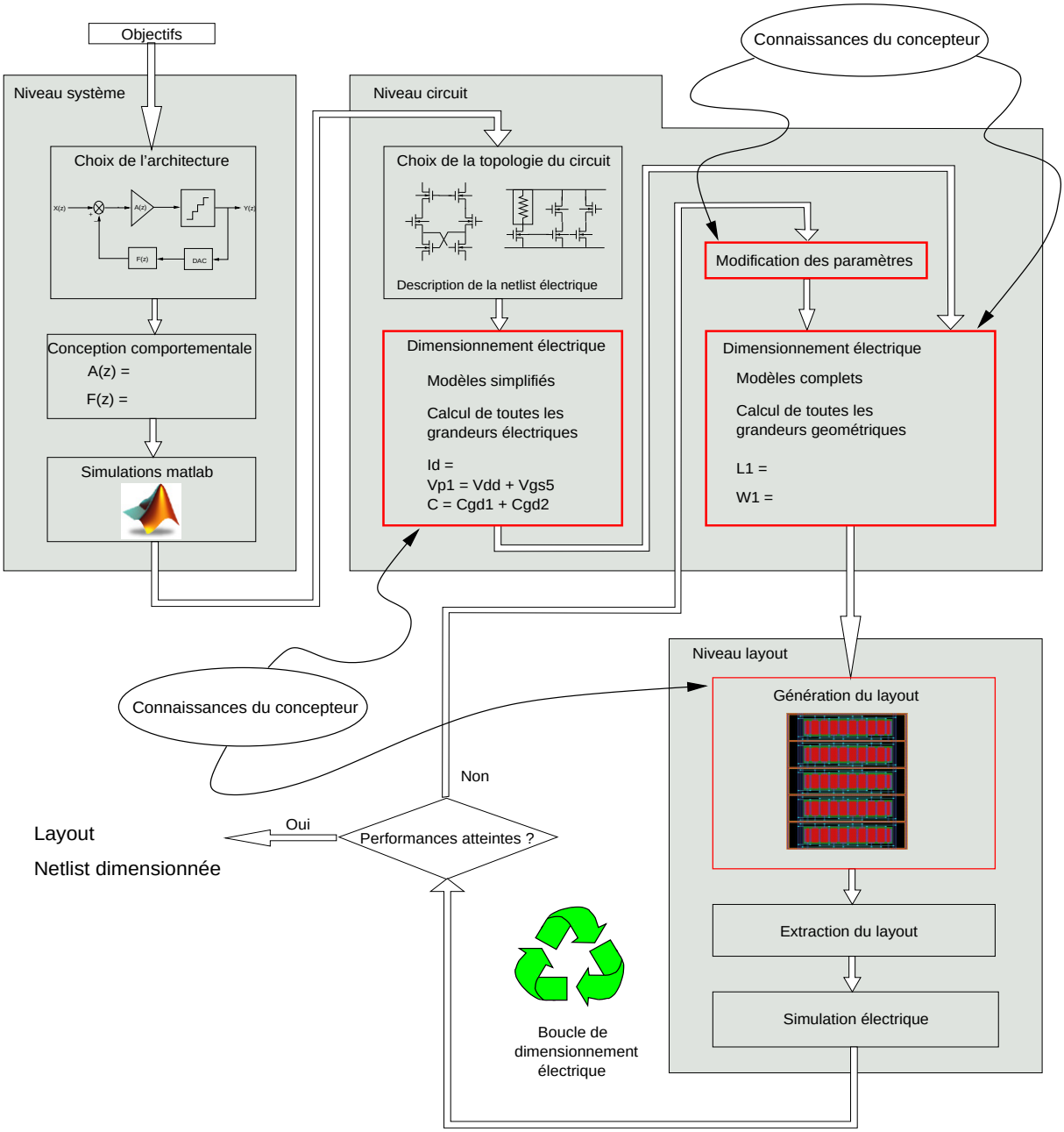


FIG. 2.7: Flot de conception classique

- L'extraction des valeurs des éléments parasites est réalisée par des outils complexes qui génèrent une grande masse de données.
- La simulation électrique est plus ou moins coûteuse en temps suivant que l'on traite le domaine fréquentiel ou temporel.

Trois conclusions se dégagent de ce schéma :

1. Le dimensionnement électrique est le point critique. Il est totalement manuel ou est

effectué par des outils dédiés [JGSV90][MPVH95]. Dans le premier cas, l'expérience n'est pas conservée, dans le second, le concepteur analogicien a peu de latitude d'intervention.

2. La qualité de la génération du dessin des masques influe grandement sur les performances résultantes. Beaucoup d'études visant à automatiser la génération du dessin des masques existent, mais peu abordent la conception concurrente électrique-physique comme [VdPS+01][DL00].
3. Les simulations électriques consomment une grande part du temps de conception (une simulation peut durer de quelques secondes à plusieurs jours). S'affranchir des simulations, ou tout du moins ne garder qu'une simulation finale à fin de vérifications doit permettre une meilleure « Exploration du Domaine de Conception » ou DSE.

2.4 Conclusion

L'objectif poursuivi est de définir un langage permettant le développement de générateurs de fonctions analogiques possédant une double généricité : se conformer aux spécifications fonctionnelles et aux procédés technologiques de fabrication. On vise à capitaliser l'expertise du concepteur, c'est à dire à lui permettre de formaliser sa méthode de dimensionnement électrique et sa méthode de dessin des masques en tenant compte des capacités parasites.

Chapitre 3

État de l'Art

Sommaire

3.1 Réalisations industrielles 16
3.1.1 Virtuoso®Neocell/Virtuoso Neocircuit de Cadence® 16
3.1.2 Barcelona Design 18
3.1.3 Sagantec 19
3.2 Réalisations universitaires 19
3.2.1 OASYS 19
3.2.2 OPASYN et ANACONDA 19
3.2.3 KOAN-ANAGRAM II 20
3.2.4 AMGIE 20
3.2.5 BALLISTIC 20
3.2.6 PAD 22
3.2.7 Océane 22
3.2.8 Microwind 23
3.3 Résumé 23

Le chapitre précédent a exposé la problématique : définir un langage permettant de réaliser des générateurs de fonctions analogiques laissant une grande liberté de conception et permettant de capitaliser l'expérience. Ce chapitre recense les principaux outils (commerciaux ou universitaires).

3.1 Réalisations industrielles

3.1.1 Virtuoso®Neocell/Virtuoso Neocircuit de Cadence®

Neocell et Neocircuit sont apparus en 1998 sous l'égide de la société Neo Linear® — rachetée depuis par Cadence — et sont intégrés à la chaîne analogique Virtuoso. Virtuoso Neocircuit est dédié au dimensionnement électrique et à l'optimisation, tandis que Virtuoso Neocell se consacre au placement/routage. Tous deux proposent des bibliothèques de cellules.

Virtuoso Neocircuit

Le flot de conception de Virtuoso Neocircuit se compose de quatre étapes (figure 3.1 [1]) :

1. Le concepteur choisit sa topologie et la saisit *via* l'interface graphique de Virtuoso. Le schéma non dimensionné peut être partiellement dimensionné.
2. Toujours à l'aide de l'interface graphique, il précise les contraintes simples comme l'appariement ou les rapports de dimensionnement.
3. Les paramètres électriques (produit gain-bande, marge de phase etc.) sont précisés avec leurs domaines de variation admissibles (contraintes). Les buts à atteindre sont aussi définis (objectifs).
4. Virtuoso se charge d'optimiser le circuit par recuit simulé en faisant appel à un simulateur externe. Le domaine de conception est ainsi exploré de façon automatique mais aveugle.

Même si le concepteur a la possibilité de saisir un certain nombre de contraintes électriques, il ne lui est pas possible d'intervenir sur la méthode d'optimisation, ni d'implanter sa propre méthode d'optimisation. L'appel à un simulateur électrique extérieur, de même que la nature des algorithmes d'optimisation, rendent la conception extrêmement chronophage.

[1] Source Cadence http ://www.cadence.com

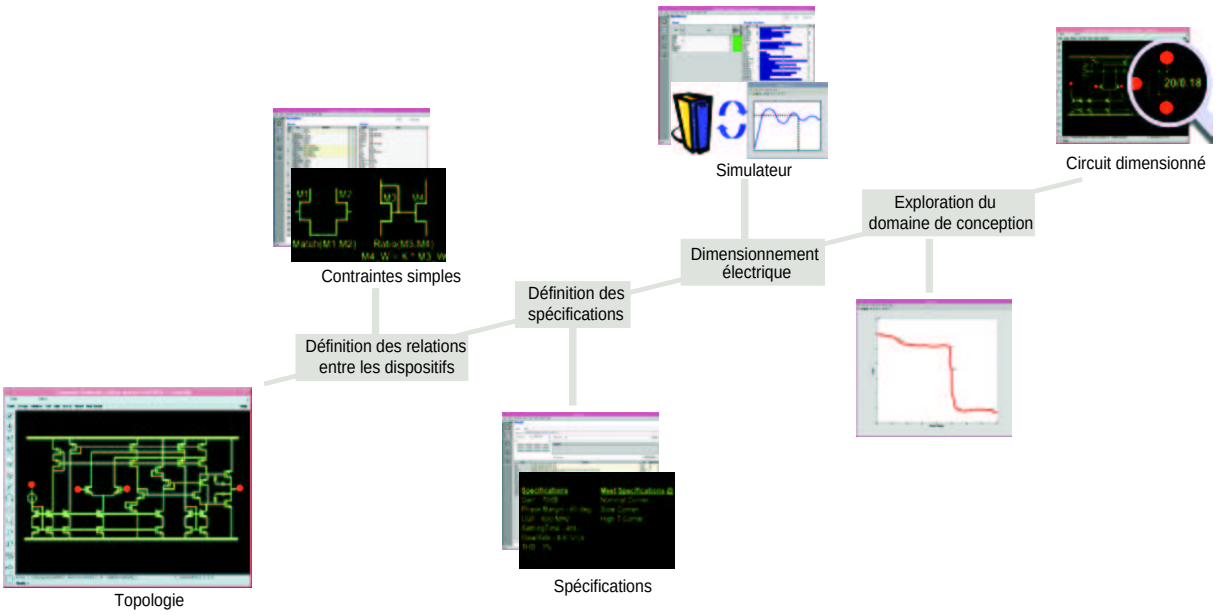


FIG. 3.1: Flot de conception Virtuoso Neocircuit [Cad]

Virtuoso Neocell

Virtuoso Neocell permet de saisir graphiquement les contraintes physiques analogiques et de placer et router le circuit. Il est à noter que les outils Neocircuit et Neocell ne collaborent pas, il n'y a aucune interaction entre les outils de dimensionnement électrique et le dessin des masques (figure 3.2).

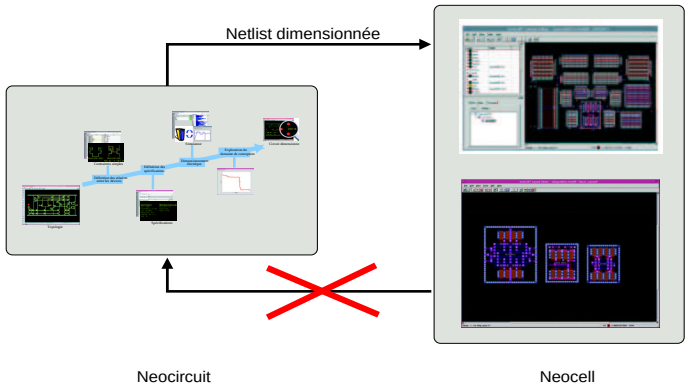


FIG. 3.2: Flot de conception Virtuoso Neocell [Cad]

Neocircuit et Neocell sont deux outils difficiles à utiliser. Le fait que les deux outils ne communiquent pas interdit la co-conception électrique-physique. L'existence de bibliothèques de cellules ne compense pas l'absence de langage dédié permettant la description de l'optimisation électrique ; ces générateurs sont d'ailleurs développés pour une technologie donnée. On est donc loin des objectifs de portabilité et de réutilisabilité.

3.1.2 Barcelona Design

Barcelona Design proposait une chaîne de conception essentiellement orientée vers les aspect système et circuit au détriment du dessin des masques. Le flot de conception est représenté figure 3.3 [2].

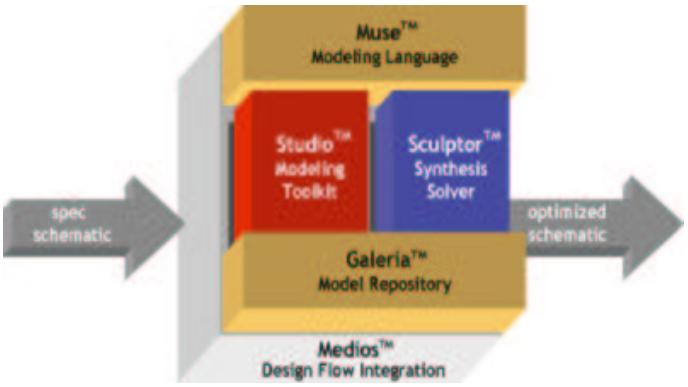


FIG. 3.3: Flot de conception Barcelona Design [Des]

1. Le schéma non dimensionné est saisi à l'aide d'un éditeur tiers, puis importé dans Studio™Modeling Toolkit. Studio accepte aussi les cellules de Galleria™et les parasites extraits.
2. Selon la dénomination de Barcelona, le « comportement » du circuit est saisi par le concepteur sous forme d'équations analytiques. Il s'agit en fait d'équations simples (marge de phase, produit gain-bande...) et non d'une description fine du comportement du circuit, la véritable description analytique étant « cachée » dans le générateur. Elle est pour l'instant inaccessible au concepteur, qui n'a aucune prise sur elle.
3. Sculptor™Synthesis Solver se charge de l'optimisation électrique par programmation géométrique, les contraintes ayant été préalablement imposées.

L'optimisation peut prendre en compte certaines contraintes géométriques (facteur de forme), ce qui suppose une estimation de la surface, mais aucun dessin des masques n'est produit lors de cette étape. Le langage de modélisation Muse™n'a jamais été rendu public. Il aurait dû permettre au concepteur analogicien d'écrire lui-même les équations analytiques de son circuit au lieu de se reposer uniquement sur la bibliothèque de générateurs fournie.

Barcelona Design proposait donc un ensemble de générateurs faciles à utiliser, mais pêchait par son manque de couplage avec le dessin des masques. La description analytique fine des éléments du circuit était impossible à réaliser faute de langage disponible. Enfin, les générateurs étaient liés à un procédé technologique.

Barcelona Design a procédé à la cessation de ses activités en mars 2005. Les quelques blocs IP développés (PLL, AOP, convertisseurs A/N et N/A) étaient très difficiles à mettre

[2]Source Barcelona Design http ://www.barcelonadesign.com

au point du fait de la variation des procédés technologiques, de l'aveu même de M. Joe Costello, président de Barcelona[3]. Il est peu probable que le langage Muse ne soit rendu public.

3.1.3 Sagantec

Sagantec propose deux produits destinés principalement à la conception analogique :

- **Anaconda** est un module pour les éditeurs Virtuoso Schematic et Virtuoso XL de Cadence. Il permet de saisir des contraintes analogiques directement sur la vue schématique et sur le dessin des masques. Il s'agit d'apporter un contrôle supplémentaire sur le dessin des masques accélérant la conception, mais non de créer des générateurs réutilisables.
- **SiClone** autorise la migration technologique de dessin des masques sous réserve de posséder dans deux technologies différentes un schéma déjà dimensionné. L'outil se charge des transformations géométriques physiques, mais la phase de redimensionnement électrique reste entièrement à la charge du concepteur.

Sagantec présente des outils destinés à améliorer la productivité du concepteur, sans néanmoins permettre la conception de générateurs.

3.2 Réalisations universitaires

3.2.1 OASYS

OASYS [HRC89] [IF90] repose sur la capitalisation des connaissances dans un outil de type système-expert. Cela nécessite la formalisation de ces connaissances (plan de dimensionnement, ou règles de dimensionnement) et l'utilisation d'un grand nombre d'heuristiques. Une telle approche donne rapidement des résultats une fois le plan de dimensionnement établi, mais la flexibilité est faible : changer les règles pour s'adapter à de nouvelles spécifications n'est pas chose aisée. Le problème de la migration technologique n'est pas pris en charge.

3.2.2 OPASYN et ANACONDA

OPASYN [KSG90] et ANACONDA [PKR+99] font partie des outils reposant sur des techniques d'optimisation numériques qui autorisent un meilleur contrôle et un codage plus rapide au détriment d'un temps d'exécution plus grand. Elles mettent en œuvre une boucle dans laquelle les objectifs définis par le concepteur relativement aux spécifications doivent être minimisés. Deux moyens sont utilisées pour évaluer les performances du circuit :

[3]Source EETIMES.fr, http ://www.eetimes.fr/ed/news/showArticle.jhtml ?articleID=60406946

- Un ensemble d'équations simplifiées modélise les caractéristiques du circuit. Cette méthode est rapide mais peu précise pour des modèles qui s'éloignent du modèle petit-signal des transistors. C'est le cas d'OPASYN.
- Des simulations électriques assurent la caractérisation du circuit. La précision est élevée (précision égale à celle du simulateur), mais le temps d'exécution est prohibitif. De nombreuses contraintes de conception (contraintes de stabilité) doivent aussi être prises en compte pour assurer un fonctionnement correct du circuit. ANACONDA fait partie de cette catégorie.

3.2.3 KOAN-ANAGRAM II

KOAN-ANAGRAM II [CGRC91] [LGS99] est dédié à la génération optimisée de dessin des masques mais ne prend pas en charge le choix du schéma ni les optimisations de niveau système. Le dessin des masques est produit par des générateurs, le dimensionnement électrique est réalisée par analyse de sensibilité du circuit et repose sur un algorithme de type recuit simulé. Les temps de calcul sont donc conséquents.

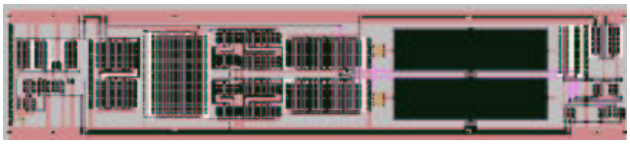
3.2.4 AMGIE

AMGIE [dPDL+01] rassemble la conception des vues système et dessin des masques dans une plateforme homogène. La méthodologie est hiérarchique avec un raffinement descendant et indépendant de la technologie cible. Il est aussi possible au concepteur de choisir la topologie des différents blocs et d'appliquer des contraintes électriques (avoir un temps d'établissement de 10 μs avec une marge de 0,1 % par exemple). Le dessin des masques est issu de générateurs.

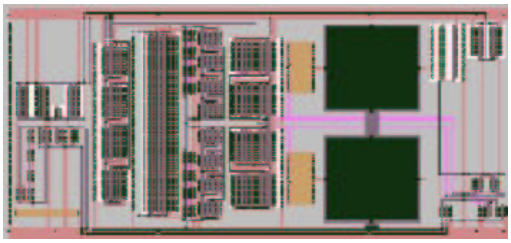
Cependant, les interactions entre le dessin des masques et le dimensionnement électrique — présentes, contrairement à Neocell — sont réalisées en deux étapes distinctes : chaque tour de la boucle d'optimisation impose une extraction et une simulation électrique. Seuls les éléments de la bibliothèque d'AMGIE peuvent être utilisés pour le circuit. Bien qu'il soit possible d'étendre cette bibliothèque (qui comprend la bibliothèque de cellules proprement dite avec toutes les informations nécessaires à la sélection de la topologie, le dimensionnement électrique et l'optimisation, et la bibliothèque technologique qui rassemble toutes les informations technologiques), il n'apparaît pas clairement que cela puisse être réalisé aisément.

3.2.5 BALLISTIC

Ballistic [ODJ+95] [OM95] est un langage de génération de dessin des masques s'interfaçant avec GDT de Mentor Graphics. De nombreuses primitives sont fournies (transistors, résistances, capacités, inductances...). Bien que ne prenant pas en compte la partie dimensionnement électrique, Ballistic est capable de réaliser automatiquement un certain nombre d'opérations nécessaires à la migration technologique comme le routage automatique avec respect de contraintes, le repliement des transistors *etc.* La figure 3.4 donne un



Hauteur de 100 μm



Hauteur de 175 μm

FIG. 3.4: Amplificateur opérationnel dans la même technologie pour deux hauteurs de cellules différentes, dessin des masques réalisé par Ballistic [ODJ+95] [OM95]

exemple de dessin des masques obtenu pour un amplificateur opérationnel réalisé avec les mêmes spécifications électriques en utilisant deux hauteurs de cellules différentes.

Le code suivant est un exemple tiré du manuel d'utilisation. Il s'agit d'un AOP (plus simple que celui de la figure 3.4, composé d'une paire différentielle, d'un miroir de courant et d'un transistor simple) dont les dimensions des transistors ont été calculées au préalable :

```
DEFAULTS {
  DPWidth = 200
  DPLength = 2*LAMBDA
  PRINT amplifier.ps
  OPTIONS GLOBAL HEIGHT=100
}

DECLARATIONS {
  DIFFPAIR qdp
  TRANSISTOR qload qbias

  SUBCIRCUIT xloads
  INSTANCE input load1 load2 loads bias
  CELL amplifier
}

BLOCKS {
  qdp TYPE=N WIDTH=DPWidth LENGTH=DPLength ASPECT=0.5 ORIENT=HOR
  qload TYPE=P WIDTH=100 UNITWIDTH=20 ASPECT=0.3 ORIENT=VER
  qbias TYPE=N WIDTH=10 LENGTH=2.4 ORIENT=HOR
}

SUBCIRCUITS {
  xloads {
    INSTANCE load1 load2 OF qload
    MOVE load2 RIGHTOF load1 SEP=10 FLIP=Y
    WIRE load1.source load2.source WIDTH=20 HTRUNC=YES TTRUNC=YES
    WIRE load1.gate load2.gate
    WIRE load1.gate load1.drain
    EXPORT in1=load1.drain in2=load2.drain
    EXPORT Vdd=load1.source X=(load1.LEFT+load2.RIGHT)/2
  }
}
```

```
CELLS amplifier {
  OPTIONS BUS TYPE=LINE SEP=100 WIDTH=10*LAMBDA
  INSTANCE loads OF xloads
  INSTANCE bias OF qbias
  INSTANCE input OF qdp
  MOVE bias ABOVE Gnd
  MOVE loads BELOW Vdd
  MOVE input BELOW loads
  WIRE input.output1 loads.in1
  WIRE input.output2 loads.in2
  WIRE input.source bias.drain SHAPE=VH
  WIRE bias.source Gnd
}
```

Ballistic n'est plus maintenu depuis 1996.

3.2.6 PAD

PAD [Kay03] est un outil interactif de conception de circuits au niveau transistor. Il repose sur une approche *knowledge based* et n'adresse que des structures simples, qui vont du transistor à l'OTA. Les procédures de dimensionnement semblent riches, mais le dessin des masques produit est individuel, dans le sens où il faudra disposer d'un outil de placement-routage pour assembler le circuit entier à partir de ses constituants. Il n'y a pas de langage permettant d'introduire un nouveau composant, il n'est pas non plus possible de définir soi-même sa propre procédure de dimensionnement.

3.2.7 Océane

Océane [Por05] a été développé à l'ENST, puis au LIP6 par Jacky Porte. L'auteur en donne la description suivante sur son site web : « Océane est destiné à la conception complète de filtres temps continu (RC) et temps discret (capacités commutées) partant d'une description "haut niveau" (gabarit) et allant jusqu'à la netlist au format SPICE constituée de transistors MOS (AOP, commutateurs...), résistances intégrées (macromodélisées) et condensateurs intégrés (macromodélisés). [D'une façon générale, Océane est] une "boîte à outils" composée d'outils de premier niveau appelés compilateurs regroupant des dispositifs de même niveau hiérarchique et d'environnements regroupant les outils spécifiques à la réalisation d'un bloc fonctionnel. Il est conçu pour recevoir en entrée un jeu de spécifications de haut niveau portant sur un bloc fonctionnel ou une fonction élémentaire et pour sortir une netlist au format standard SPICE, correspondant à une description topologique des composants élémentaires, transistors, résistances et condensateurs, géométriquement dimensionnés et affectés de leurs paramètres de style. Un interfaçage avec d'autres outils commerciaux ou libres (simulateurs, générateurs automatiques de masques...) est ainsi directement réalisable ».

Il s'agit donc d'un outil exclusivement destiné à la synthèse électrique, disposant d'une interface graphique écrite en tcl/tk, mais dépourvu de langage de description. Il n'est cependant pas prévu de décrire sa propre procédure de dimensionnement et il est difficile de composer de nouveaux circuits à partir des éléments disponibles. L'outil est toutefois

d'une grande utilité lors de la conception d'un circuit pour les possibilités de vérifications qu'il apporte.

3.2.8 Microwind

Microwind [Sic02], dont il existe une version commerciale, est sans doute l'un des outils les plus aboutis à destination de l'enseignement de la micro-électronique. Bien que relativement fermé, il est néanmoins très complet puisque son champ d'action va de la simulation analogique (pas de simulateur externe requis) à l'édition de dessin des masques en passant par le domaine numérique avec Verilog et un simulateur logique, cela pour des technologies jusqu'en 0.18 μm. Microwind supporte les modèles 1, 3 et BSIM4.

Il manque toutefois une ouverture vers la conception de plus haut niveau (niveau système), la possibilité d'implanter ses procédures de dimensionnement, la prise en compte du VHDL et une ouverture vers la simulation mixte. Il n'est pas possible de générer un dessin des masques ; en fait, il n'existe pas de générateurs. Microwind est plutôt une juxtaposition de modules (éditeur de dessin des masques, extracteur, simulateur analogique, simulateur numérique, éditeur de schémas) permettant d'acquérir des connaissances en micro-électronique et d'expérimenter sans pour autant posséder les coûteux outils commerciaux industriels.

3.3 Résumé

Les outils cités adressent une ou plusieurs étapes de la conception analogique : le niveau système, le niveau circuit et le niveau dessin des masques.

Le tableau 3.1 récapitule les différentes possibilités des outils cités. Peu d'outils sont capables d'adresser à la fois la synthèse électrique et la synthèse physique. Lorsque c'est le cas, le couplage est faible ou la personnalisation des procédures de dimensionnement est impossible. La capitalisation des connaissances souffre d'une absence de méthodologie : il est très difficile de réutiliser l'expertise mise dans la conception d'un circuit. La plupart du temps, le retour sur investissement est faible, puisque les projets futurs ne peuvent en bénéficier. La conception est souvent longue en raison de l'utilisation d'un simulateur externe.

Les faiblesses relevées ici constituent les objectifs que CAIRO+ doit relever. Les multiples facteurs (rapidité d'exécution, portabilité...) dont il faut tenir compte seront exposés dans le chapitre suivant ; ils sont les conditions nécessaires à la réalisation d'un générateur portable.

TAB. 3.1: Comparaison des possibilités des outils présentés

				Virtuoso Neocircuit	Virtuoso Neocell	Barcelona Design	Sagantec	OASYS	OPASYN	ANACONDA	KOAN-ANAGRAM II	AMGIE	Ballistic
Système	Interface graphique			-	-	●	-	-	-	-	-	-	-
	Langage de modélisation			-	-	-	-	-	-	-	-	-	-
	Équations simples			-	-	●	-	-	-	-	-	-	-
	Équations analytiques			-	-	●	-	-	-	-	-	-	-
	Bibliothèque			-	-	●	-	-	-	-	-	●	-
Circuit	Interface graphique			●	-	●	●	-	-	-	-	-	-
	Langage de modélisation			-	-	-	-	-	-	-	-	-	-
	Description analytique			-	-	●	-	-	●	●	●	-	-
	Bibliothèque			●	-	●	●	●	?	?	-	●	-
	Extension possible			●	-	-	●	●	●	●	●	●	-
	Dim. électrique	**Optimisation**	Système expert	-	-	-	-	●	-	-	-	-	-
			numérique	-	-	-	-	-	●	-	-	-	-
			Knowledge based	-	-	●	-	-	-	-	-	-	-
			Simulateur externe	●	-	-	-	-	-	●	-	●	-
			Personnalisable	-	-	-	-	●	?	?	-	-	-
		Contraintes		●	-	●	-	●	?	?	●	●	-
		Objectifs		●	-	●	-	●	?	?	●	●	-
Layout	Interface graphique			-	●	-	●	-	-	-	-	-	●
	Langage de description			-	-	-	-	-	-	-	-	-	●
	Placement			-	●	-	●	-	●	-	●	?	●
	Routage			-	●	-	●	-	-	-	●	?	●
	Contraintes physiques			-	●	-	●	-	-	-	●	?	●
Général	*Codesign* physique-électrique			-	-	-	-	-	-	-	●	●	-
	Utilisation de générateurs			●	●	-	●	-	?	?	?	●	●
	Personnalisation des générateurs			-	-	-	-	-	?	?	?	●	●
	Bibliothèque de cellules			●	●	●	●	-	?	?	?	●	●

Chapitre 4

Principes généraux du langage CAIRO+

Sommaire

4.1 Méthode de conception proposée 26
4.2 Le précurseur : le langage CAIRO 29
4.3 Schéma non dimensionné 29
4.4 Placement relatif 30
4.5 Dimensionnement électrique 32
4.6 Génération du dessin des masques 33
4.7 Prise en compte des capacités parasites 34
4.8 Migration technologique 35
4.9 Conclusion 37

Ce chapitre présente les principes généraux du langage CAIRO+. Celui-ci supporte une méthode de conception systématique s'appuyant sur quatre principes : la formalisation de l'expertise du concepteur analogicien, en particulier pour ce qui concerne le dimensionnement électrique ; l'interaction forte entre synthèse électrique et synthèse physique, permettant la prise en compte des capacités parasites ; la réutilisation des composants analogiques génériques préexistants, à travers une approche hiérarchique multiniveaux ; la double généricité des générateurs qui doivent accepter différentes spécifications fonctionnelles et différents procédés de fabrication.

4.1 Méthode de conception proposée

On appellera module un composant analogique générique et réutilisable, défini par sa topologie, c'est à dire par un schéma électrique non dimensionné où les valeurs des paramètres électriques (résistances, capacités, transconductances, etc.) ne sont pas définies. On appellera dans la suite schéma non dimensionné ce type de description. À tout module est associé un programme de génération, ou générateur, écrit en utilisant le langage CAIRO+. Un module peut instancier d'autres modules préexistants, ce qui signifie qu'un générateur CAIRO+ peut appeler d'autres générateurs CAIRO+.

On appellera dispositif élémentaire un module terminal, c'est à dire un module qui n'instancie aucun autre module. Un dispositif élémentaire correspond généralement à un composant élémentaire du schéma électrique, tel qu'un transistor MOS, une paire différentielle, une capacité ou une résistance. Les dispositifs élémentaires, comme les modules, sont des composants génériques (il existe donc un générateur associé à chaque dispositif élémentaire). Tout dispositif élémentaire est caractérisé par un petit nombre de paramètres électriques qui définissent le comportement du dispositif élémentaire : longueur L et largeur W du canal (en micromètres) pour un transistor MOS, valeur C (en farad) pour une capacité, valeur R (en ohm) pour une résistance.

On appellera module de premier niveau un module qui n'instancie que des dispositifs élémentaires. La figure 4.1 tirée de [IF90] (page 107) donne un exemple de module de premier niveau et illustre sa décomposition en dispositifs élémentaires.

L'exécution d'un programme générateur d'un module M, pour une spécification fonctionnelle donnée et pour un procédé de fabrication cible, va générer d'une part le fichier contenant un schéma électrique détaillé (qui peut par exemple être simulé sous SPICE), et d'autre part le fichier contenant le dessin des masques pour le procédé de fabrication cible. Si le module M instancie lui-même d'autres modules (c'est à dire si M n'est pas un module de premier niveau), le schéma généré, ainsi que le dessin des masques seront

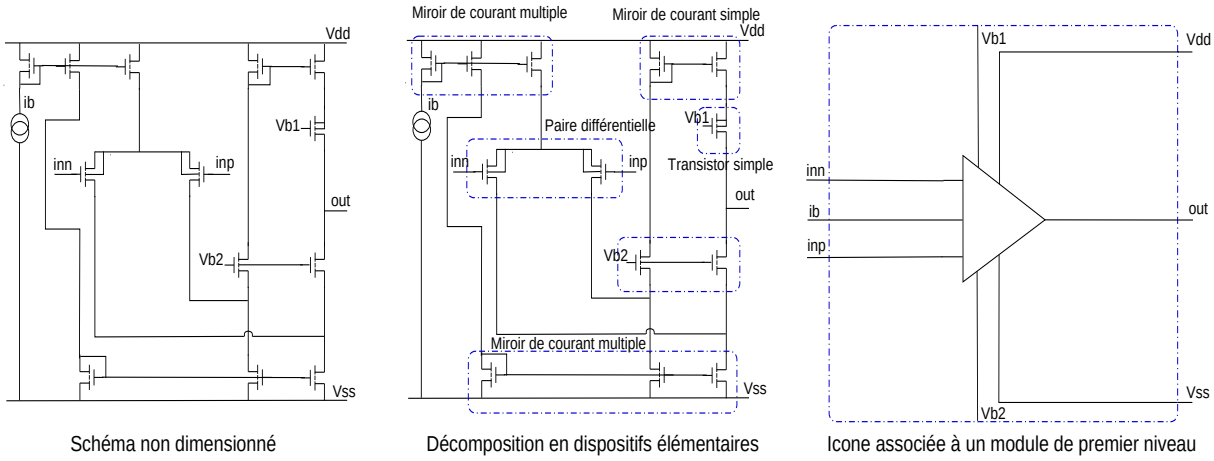


FIG. 4.1: Décomposition en dispositifs élémentaires et icône associée à un amplificateur opérationnel cascode replié à transconductance (OTA) selon [IF90] (page 107)

hiérarchiques.

L'exécution du programme de génération d'un module M écrit avec le langage CAIRO+ comporte obligatoirement quatre phases, décrites dans la figure 4.2. Ces quatre phases sont détaillées dans la suite de ce chapitre, mais nous en donnons ci-dessous une première présentation générale :

- La phase CREATE correspond à la création en mémoire des structures de données représentant d'une part le schéma du module M, d'autre part le placement relatif des modules instanciés par M. On en profite pour déclarer les paramètres électriques du module qui seront utilisés par la phase de dimensionnement électrique (pour un amplificateur OTA, les paramètres principaux sont, par exemple, le gain, la fréquence de coupure, la marge de phase, la dynamique de sortie et le « slew rate »). Si le module M n'est pas un module de premier niveau (c'est à dire si le générateur de M fait appel à des générateurs de modules qui ne sont pas des dispositifs élémentaires), la construction est récursive et les structures de données construites en mémoire ont une structure hiérarchique multiniveaux. On construit donc explicitement l'arborescence de tous les modules et sous-modules instanciés, jusqu'aux dispositifs élémentaires terminaux.
  À la fin de cette première phase, on a donc construit en mémoire un schéma hiérarchique multiniveaux non dimensionné (les valeurs des paramètres électriques des dispositifs élémentaires ne sont pas définies), et un placement relatif (seules les positions relatives des modules et des dispositifs élémentaires sont définies, mais les surfaces occupées par chaque dispositif élémentaire et ses coordonnées ne sont pas connues).
- La phase DSES (Design Space Exploration and Sizing) correspond à la phase de dimensionnement électrique des dispositifs élémentaires. Dans cette phase, le concepteur du module M doit explicitement utiliser les procédures de dimensionnement des différents sous-modules instanciés par M en fonction des spécifications fonction-

nelles souhaitées pour cette réalisation particulière du module M. Il existe en fait plusieurs fonctions de dimensionnement associées à un même module suivant les spécifications fonctionnelles recherchées.
À la fin de cette seconde phase, tous les dispositifs élémentaires instanciés sont dimensionnés, pour le procédé de fabrication cible, ou bien le générateur du module M renvoie un code d'erreur expliquant pourquoi les spécifications fonctionnelles demandées ne sont pas compatibles avec le procédé de fabrication sélectionné.

- La phase PLACE correspond à la génération effective du dessin des masques de chacun des dispositifs élémentaires instanciés dans l'arborescence de modules dont le module M est la racine, et au placement définitif des dispositifs élémentaires les uns par rapport aux autres.
  À la fin de cette troisième phase, tous les dispositifs élémentaires instanciés dans le module M sont placés, mais ne sont pas physiquement interconnectés. On a bien sûr pris soin de réserver entre les dispositifs élémentaires des espaces réservés pour tracer les fils de routage.
- La phase ROUTE correspond au tracé des fils d'interconnexion entre les dispositifs élémentaires.
  À la fin de cette quatrième phase, le dessin des masques de tous les modules présents dans l'arbre de modules dont M est la racine est disponible.

Comme cela apparaît dans la figure 4.2, la prise en compte des capacités parasites est rendue possible par une boucle permettant d'itérer sur les trois phases DSES, PLACE et ROUTE. Ce mécanisme est décrit au §4.7.

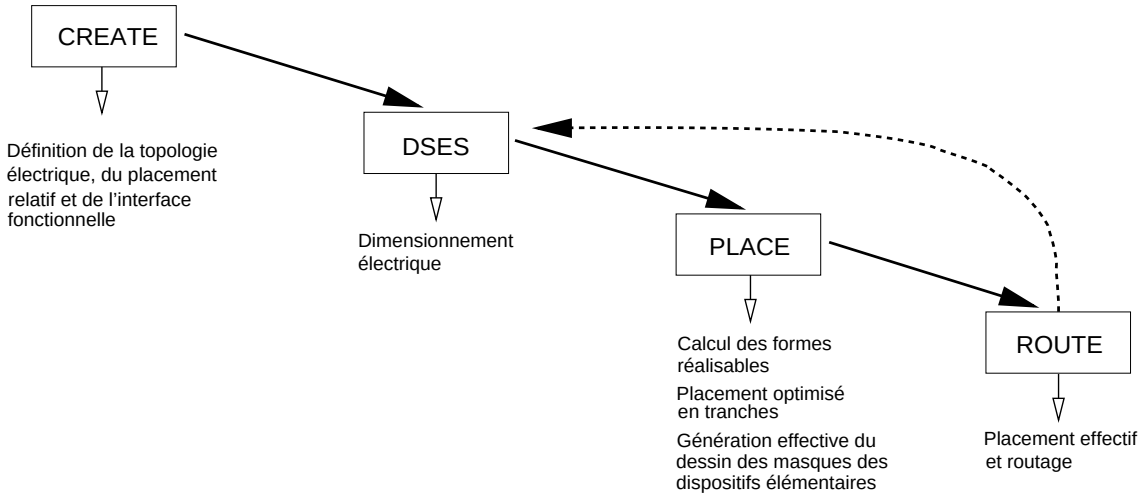


FIG. 4.2: Étapes de conception dans le flot CAIRO+

4.2 Le précurseur : le langage CAIRO

Avant de détailler les services fournis par le langage CAIRO+ pour chacune des quatre phases décrites ci-dessus, il convient de replacer CAIRO+ dans l'historique du projet CAIRO au laboratoire LIP6. Une première tentative de définition d'un langage de conception de fonctions analogiques génériques a été présentée dans le cadre de la thèse de Mohamed Dessouky[Des01]. Beaucoup des objectifs définis ci-dessus étaient déjà présents dans le projet CAIRO. Le langage CAIRO était constitué d'un ensemble de fonctions C.

Les générateurs de CAIRO ne prenaient pas en charge le dimensionnement électrique, qui était réalisé en externe par un outil tiers (Comdiac, maintenant Océane). Il n'y avait donc pas de procédures de dimensionnement. Le seul mécanisme de communication hiérarchique existant ne permettait qu'une communication descendante. L'unique fonction des générateurs de CAIRO était de générer le dessin des masques, lequel reposait déjà sur le concept de placement relatif. La généralisation a été apportée par CAIRO+ et le concept de conteneurs. Cette généralisation a permis la formalisation de la fonction de forme, fonction de forme dont la définition initiale conduisait à un placement sous-optimal. L'algorithme d'optimisation du placement pouvait aussi ne pas converger, contrairement à la formulation analytique actuelle.

CAIRO+ a aussi apporté, avec l'exploration du domaine de conception et le dimensionnement électrique, la calculette transistor, qui assure l'indépendance technologique d'un point de vue électrique en incluant directement le modèle électrique (BSIM3). Un ensemble de fonctions pour l'évaluation des capacités parasites de routage ou capacités parasites des dispositifs élémentaires améliore la qualité du dimensionnement électrique.

4.3 Schéma non dimensionné

Un module M est défini par son schéma non dimensionné (en anglais : « netlist template »), et non par sa fonctionnalité. Deux amplificateurs opérationnels réalisant la même fonction mais possédant des structures internes différentes correspondent à des modules différents, et seront implantés par deux générateurs indépendants.

Un module peut instancier des composants élémentaires (les dispositifs élémentaires), mais il peut également instancier des composants complexes (c'est à dire d'autres modules). Le schéma non dimensionné d'un module M possède donc une structure hiérarchique bien connue des concepteurs de circuits numériques : l'arbre d'instanciation possède une profondeur quelconque, les feuilles sont les dispositifs élémentaires, la racine est le module M, et les nœuds intermédiaires sont les sous-modules instanciés. Comme dans toute méthode de conception hiérarchique, le concepteur du module M doit décrire la structure interne du module M, mais n'a évidemment pas besoin de décrire la structure interne des modules instanciés par M.

Dans un schéma non dimensionné, les valeurs des paramètres électriques des dispositifs élémentaires instanciés dans ce module (en particulier les dimensions des transistors ou les valeurs de résistances et capacités) ne sont pas définies. Par voie de conséquence, les valeurs des paramètres électriques du module (par exemple la valeur du gain, ou la marge de phase

pour un amplificateur) ne sont pas définies non plus. Le schéma non dimensionné est défini dans la première partie du programme de génération (phase CREATE), et c'est l'étape suivante (phase de dimensionnement électrique DSES) qui est chargée de donner des valeurs aux paramètres électriques des dispositifs élémentaires instanciés par le module, en fonction des spécifications fonctionnelles souhaitées, et du procédé de fabrication cible (figure 4.3 et 4.4) .

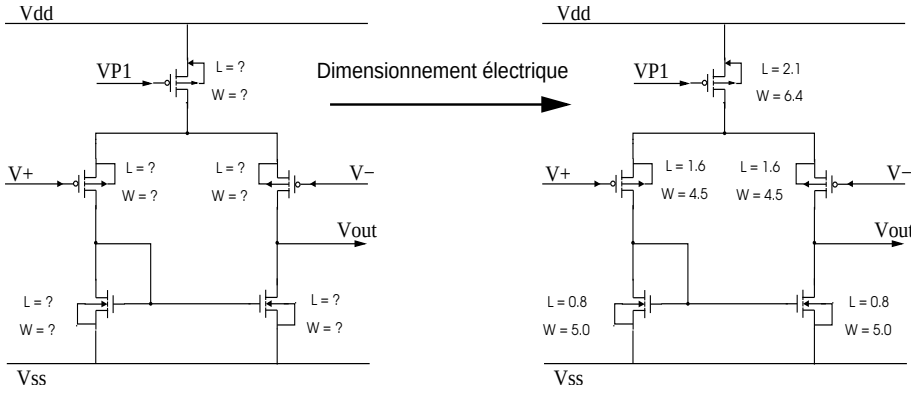


FIG. 4.3: Schéma électrique d'un OTA de type P

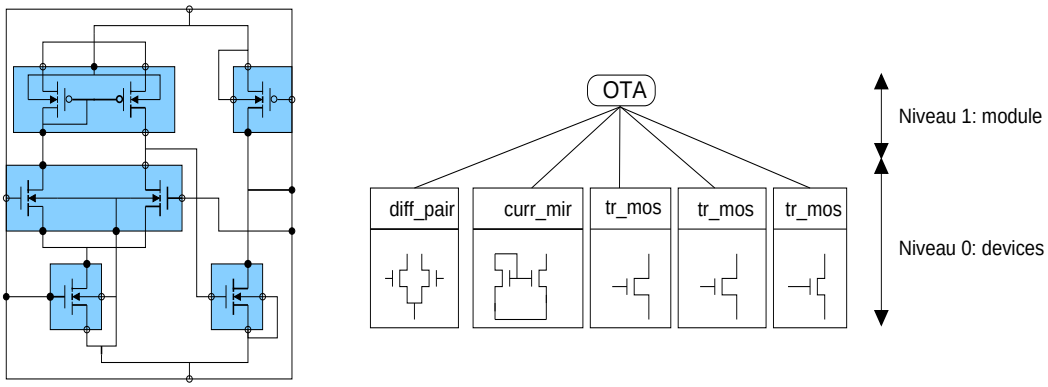


FIG. 4.4: Arbre des modules pour un OTA Miller

Il est important de noter ici que le concepteur analogicien qui souhaite écrire un générateur pour un nouveau module (c'est à dire pour un nouveau schéma), tout en réutilisant des générateurs de modules préexistants, n'a pas besoin de re-décrire la structure interne des sous-modules instanciés.

4.4 Placement relatif

À la différence d'autres environnements de conception de circuits analogiques, le langage CAIRO+ met un fort accent sur la génération conjointe du schéma électrique et du

dessin des masques. En effet, la réalisation physique du circuit introduit des éléments parasites dépendant du dessin des masques (telles que les capacités de drain des transistors et les capacités d'interconnexion), qui ont une forte influence sur les performances du circuit final.

Pour permettre au concepteur d'un générateur d'un module M de contrôler explicitement le plan de masse du module, CAIRO+ ne fournit pas de fonctions de placement automatique. En revanche, CAIRO+ fournit au concepteur des fonctions lui permettant de décrire explicitement le placement relatif (en anglais : « layout template ») des objets instanciés par le module M (dispositifs élémentaires ou autres modules) les uns par rapport aux autres. Puisque le placement relatif doit être défini pour tout module, cela est vrai pour les sous-modules instanciés par M. Le concepteur n'a donc pas à redéfinir le placement relatif des objets contenus dans les sous-modules de niveau (n-1) instanciés par M.

La technique utilisée pour définir le placement relatif des objets instanciés dans un module est la technique du placement en tranches, ou *slicing tree* [DW86][TLG04b]. Cette technique permet de structurer la surface occupée par le module en une série de zones appelées conteneurs. Un conteneur contient un nombre variable d'autres conteneurs empilés verticalement ou horizontalement. Les objets placés dans les conteneurs terminaux (c'est à dire les conteneurs qui ne contiennent pas d'autres conteneurs) sont les dispositifs élémentaires ou modules instanciés par le module M. Le nombre de conteneurs terminaux est égal au nombre d'objets instanciés par le module M. Cette structuration peut être décrite par un arbre des conteneurs (figure 4.5), qu'il ne faut pas confondre avec l'arbre d'instanciation des modules. Il existe donc un arbre des conteneurs pour chaque module qui n'est pas un dispositif élémentaire. Ce type de description est bien adapté lorsqu'on cherche à décrire les positions relatives d'objets dont on ne connaît a priori ni la forme ni la surface.

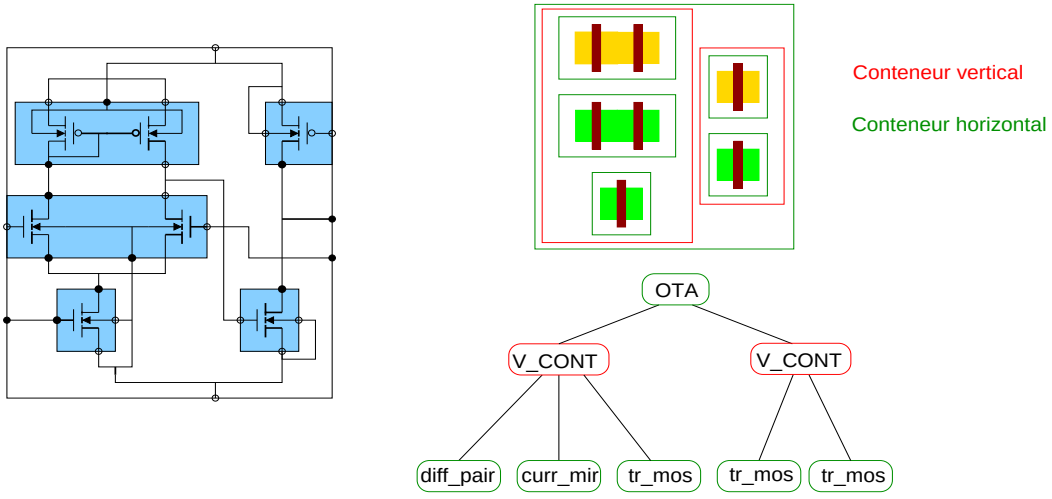


FIG. 4.5: Arbre des conteneurs pour un OTA Miller

Ce placement relatif est défini dans la première partie du générateur (phase CREATE).

4.5 Dimensionnement électrique

Les services fournis par le langage CAIRO+ pour cette phase de dimensionnement électrique (DSES) sont présentés en détail dans le chapitre 5. Nous ne présentons ici que les principes généraux.

Pour un module M, le code des différentes procédures de dimensionnement correspondant à la phase DSES est évidemment spécifique, puisque c'est cette partie du code qui formalise l'expertise du concepteur analogicien. Le but principal d'une procédure de dimensionnement d'un module M est de fixer les valeurs des paramètres électriques (dimensions des transistors MOS, valeurs des éventuelles résistances, inductances ou capacités) de tous les dispositifs élémentaires qui sont les feuilles de l'arbre de modules dont M est la racine.

Notons que la méthode de dimensionnement est hiérarchique : dans le cas où M n'est pas un module de premier niveau (il contient des modules autres que des dispositifs élémentaires), la procédure de dimensionnement de M ne manipule pas directement les dispositifs élémentaires qui ne lui appartiennent pas, mais doit utiliser les procédures de dimensionnement des modules M' instanciés par M.

La procédure de dimensionnement d'un module M doit pouvoir être utilisée quel que soit le procédé de fabrication choisi, ce qui introduit une énorme incertitude sur ce qui est possible et sur ce qui ne l'est pas. Par conséquent le dimensionnement se déroule en deux étapes :

- Dans une première étape le module M interroge les différents modules M' qu'il contient pour leur demander s'ils sont capables de satisfaire une certaine spécification fonctionnelle (une valeur de gain pour un amplificateur par exemple), et à quel coût (consommation, surface, fréquence, etc.). La réponse à cette interrogation peut être négative car le procédé de fabrication n'est pas connu au moment de l'écriture du générateur, et il est clair que toute spécification fonctionnelle n'est pas réalisable par tout procédé de fabrication. Ce processus d'interrogation est hiérarchique et se propage dans l'arbre des modules de la racine M vers les feuilles que sont les dispositifs élémentaires. En effet, seuls les générateurs de dispositifs élémentaires ont connaissance du procédé de fabrication choisi et peuvent donc statuer sur la faisabilité de la spécification demandée. Les requêtes descendent de la racine M vers les dispositifs élémentaires, et les réponses remontent des dispositifs élémentaires vers la racine M.
- Dans une seconde étape, le module M choisit une solution (si elle existe) et impose une spécification définitive pour chacun des modules qui lui appartiennent. Cet ordre se propage de façon descendante vers les feuilles de l'arbre des modules et aboutit au dimensionnement effectif de tous les dispositifs élémentaires. Si la conclusion de l'étape d'interrogation est que le procédé de fabrication choisi ne permet pas de satisfaire les spécifications, le module M renvoie un code d'échec.

Tout module de niveau hiérarchique N doit fournir deux types de procédures de dimensionnement au module de niveau hiérarchique (N+1) :

- Les procédures utilisées dans l'étape d'interrogation sont appelées procédures incomplètes. Une procédure incomplète d'un module M ne fournit pas la garantie que tous les dispositifs élémentaires instanciés dans l'arbre dont M est la racine seront effectivement dimensionnés. Par ailleurs, il peut exister plusieurs procédures incomplètes pour un même module car il peut exister plusieurs formulations pour interroger un module sur ce qu'il est capable de faire.
- Les procédures utilisées dans l'étape de décision sont appelées procédures complètes. Une procédure complète d'un module M impose un jeu de spécifications fonctionnelles qui garantit que tous les dispositifs élémentaires instanciés dans l'arbre dont M est la racine seront effectivement dimensionnés. Pour cela, une procédure complète d'un module de niveau N doit toujours appeler des procédures complètes pour les modules de niveau (N-1). Comme il existe plusieurs façons d'imposer une spécification fonctionnelle à un module, il peut exister plusieurs procédures complètes pour un même module.

L'ensemble des procédures de dimensionnement d'un module M (complètes et incomplètes) constitue l'interface fonctionnelle du module M (figure 4.6)

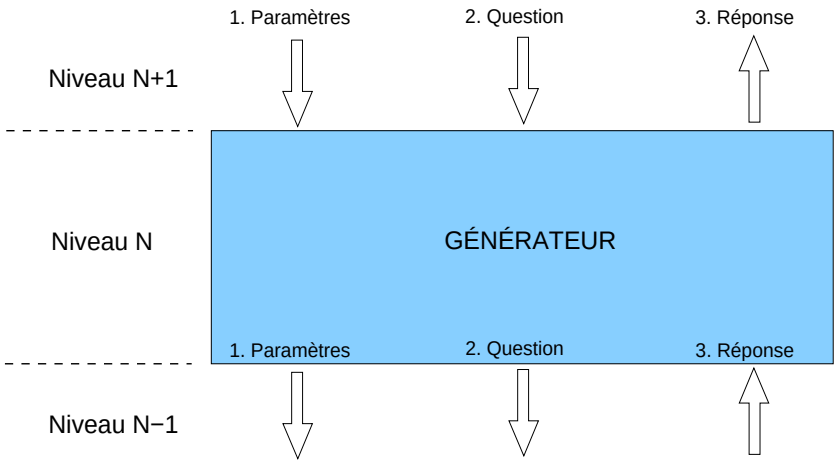


FIG. 4.6: Communication entre générateurs

4.6 Génération du dessin des masques

La génération du dessin des masques est réalisée dans les deux dernières parties du générateur (phases PLACE et ROUTE).

Le but essentiel de la phase PLACE est de générer le dessin des masques de chacun des dispositifs élémentaires instanciés et de les placer effectivement les uns par rapport aux autres en ménageant des canaux vides entre les dispositifs élémentaires pour permettre le routage ultérieur des fils d'interconnexion. La difficulté est évidemment que la surface occupée par chacun de ces dispositifs élémentaires peut varier fortement entre deux réalisations du même module et que ces variations ne vont pas forcément dans le même sens

pour les différents dispositifs élémentaires d'un même module. Les surfaces dépendent également du procédé de fabrication (qui n'est évidemment pas connu au moment où l'on écrit le programme générateur). On peut imaginer que la surface occupée par les transistors diminue lorsqu'on passe d'un procédé de fabrication CMOS 0,35 μm à un procédé CMOS 0,12 μm alors que que la surface occupée par les capacités augmente. Heureusement, on dispose d'un degré de liberté puisque, pour un dispositif élémentaire particulier et pour un dimensionnement électrique de ce dispositif élémentaire (c'est à dire pour une valeur fixée de ses paramètre électriques), il existe plusieurs réalisations possibles, et donc plusieurs formes possibles. Par exemple, pour un procédé de fabrication donné, la valeur d'une capacité est essentiellement proportionnelle à la surface des armatures, et on peut faire varier le rapport hauteur sur largeur, à surface constante, sans modifier la valeur de la capacité. Nous verrons au chapitre 6 que cette propriété de déformation à surface pratiquement constante peut se généraliser à tous les générateurs de dispositifs élémentaires disponibles dans l'environnement CAIRO+. Le problème est donc de placer des objets de surfaces variables, mais déformables, en respectant les directives de placement relatif et en minimisant la surface totale. Grâce à la technique du « slicing tree », ce problème complexe a une solution algorithmique générale qui est exposée en détail dans le chapitre 6. Par conséquent, la génération effective du placement est totalement automatique et ne nécessite de la part du concepteur du générateur aucun effort autre que les directives de placement relatif déjà définies dans la phase CREATE.

Le but essentiel de la phase ROUTE est le tracé effectif des fils d'interconnexion entre les dispositifs élémentaires. Ce tracé peut être automatique ou explicite. Dans la version actuelle de CAIRO+, il s'agit d'un routage procédural explicite (figure 4.7). Le concepteur décrit explicitement mais de façon symbolique (c'est à dire sans manipuler de coordonnées chiffrées) le cheminement des fils d'interconnexion dans les canaux définis par la structure arborescente des conteneurs. Ici encore, le routage est hiérarchique, c'est à dire que le concepteur d'un module M ne doit décrire que les fils permettant d'interconnecter les objets (dispositifs élémentaires ou modules) directement instanciés par M. Cette description explicite du routage est assez fastidieuse, et peut en principe être automatisée.

4.7 Prise en compte des capacités parasites

Les capacités parasites liées à la réalisation physique (tels que les capacités de drains des transistors ou les capacités des fils d'interconnexion) peuvent modifier significativement le comportement d'un composant analogique. Dans le flot de conception proposé, les procédures de dimensionnement (DSES) sont nécessairement exécutées avant les phases de génération du dessin des masques (PLACE et ROUTE) puisque qu'il faut avoir calculé les valeurs des paramètres électriques des dispositifs élémentaires pour pouvoir générer le dessin des masques. Mais il est nécessaire d'avoir terminé le routage pour pouvoir évaluer précisément les valeurs des capacités parasites. On a donc une situation d'interdépendance entre synthèse électrique et synthèse physique, qui peut être résolue par une boucle d'itération.

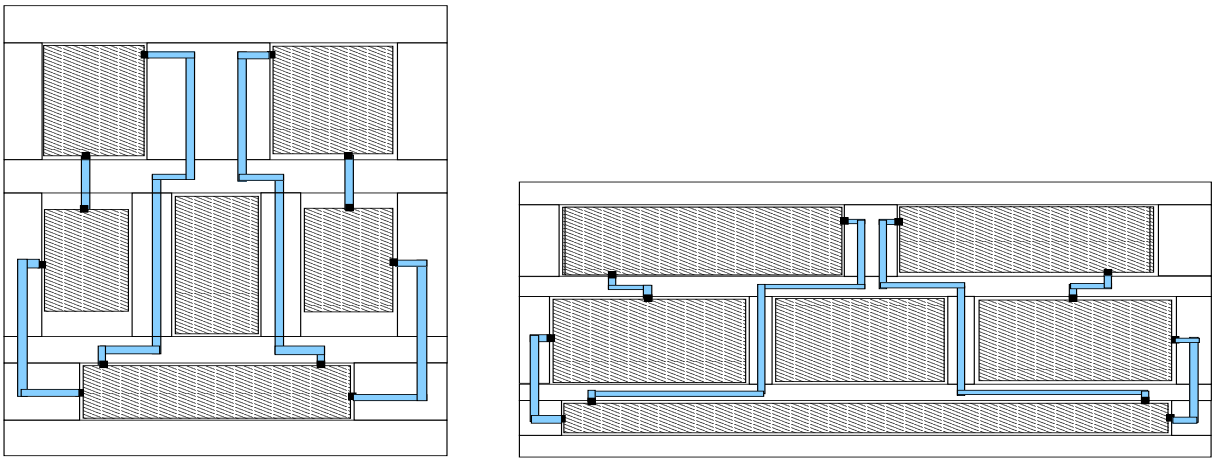


FIG. 4.7: Deux réalisations d'un même routage procédural

À la fin de la phase ROUTE, le dessin des masques est entièrement disponible en mémoire. Il est donc possible d'évaluer précisément les capacités parasites. Cette information peut être exploitée pour rétro-annoter le schéma, et relancer le calcul du dimensionnement de la phase DSES. Il suffit que les procédures de dimensionnement écrites par le concepteur du générateur de module prennent en compte systématiquement l'effet des capacités parasites, et fassent « comme si » les valeurs de ces capacités parasites étaient disponibles. Lors de la première itération, on prend des valeurs estimées par défaut (puisque le dessin n'est pas disponible), mais dès la seconde itération ces valeurs deviennent beaucoup plus exactes, et le processus converge très vite.

Ceci nécessite seulement l'existence d'une fonction d'évaluation des capacités des fils de routage (un extracteur très simplifié) dans l'environnement CAIRO+, ainsi que d'une fonction d'évaluation des capacités parasites dans les générateurs de dispositifs élémentaires.

4.8 Migration technologique

On analyse dans ce paragraphe les différentes techniques qui permettent aux générateurs écrits avec le langage CAIRO+ d'être indépendants du procédé de fabrication choisi lors de l'utilisation du générateur.

La prise en compte du procédé de fabrication intervient à trois endroits dans le flot décrit par la figure 4.2 :

- Le procédé de fabrication choisi intervient évidemment dans l'interface fonctionnelle des dispositifs élémentaires (c'est à dire dans les procédures de dimensionnement attachées à chacun des dispositifs élémentaires disponibles dans l'environnement CAIRO+). Par exemple, le dispositif élémentaire transistor NMOS fournit une fonction I_NMOS(L,W,Vgs) qui calcule (pour le procédé de fabrication sélectionné) le courant fourni par un transistor NMOS de longueur L, préalablement polarisé,

et soumis à une tension de grille Vgs. Ce dispositif élémentaire fournit également une fonction NMOS_W(I, L, Vgs), qui calcule la largeur du canal nécessaire pour qu'un transistor NMOS de longueur L, préalablement polarisé, et soumis à une tension de grille Vgs fournisse un courant I. Il est clair que les résultats renvoyés par ces fonctions dépendent fortement du procédé de fabrication choisi (c'est à dire en pratique des paramètres du modèle de transistor BSIM3 fourni par le fabricant). Le point important ici est que toutes les dépendances concernant le comportement « électrique » du transistor NMOS vis à vis du procédé de fabrication peuvent être « encapsulées » dans les fonctions définissant l'interface fonctionnelle du dispositif élémentaire NMOS. On peut appliquer cette même technique d'encapsulation pour tous les dispositifs élémentaires CAIRO+ de façon à fournir au concepteur une interface procédurale qui masque complètement le choix du procédé de fabrication dans la phase de dimensionnement électrique. De façon abrégée, on appelle calculette MOS cet ensemble de fonctions de dimensionnement associées au transistors NMOS et PMOS, qui constituent l'interface fonctionnelle des dispositifs élémentaires transistor MOS (figure 4.8). Il est à noter que cette calculette MOS (qui fait l'objet de la thèse de L. de Lamarre), fournit au concepteur des services comparables aux services rendus par un simulateur électrique de type SPICE, dans la phase de dimensionnement DSES.

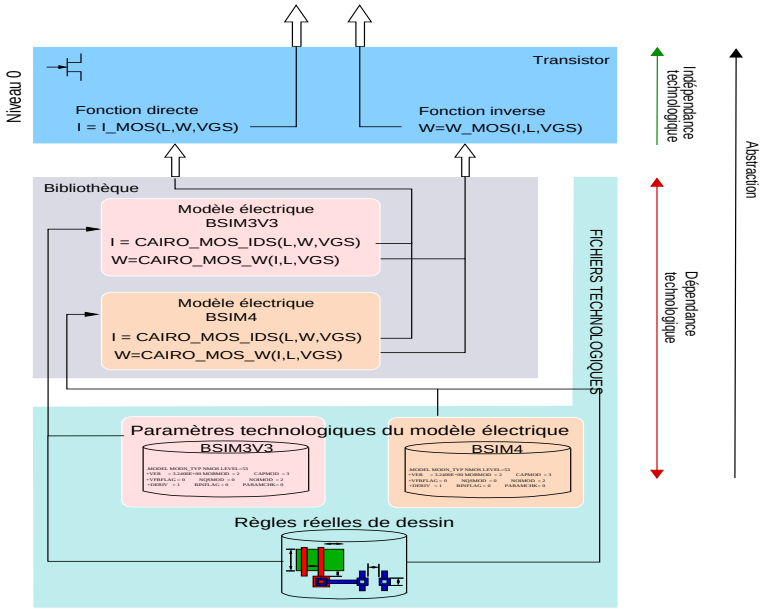


FIG. 4.8: Abstraction du modèle électrique

- Par ailleurs, le procédé de fabrication intervient dans la génération du dessin des masques des dispositifs élémentaires. Là encore, CAIRO+ permet d'encapsuler les règles de dessin associées à un procédé de fabrication particulier (largeur minimale des fils, distances minimales entre deux masques etc.) grâce à une interface procédurale relativement simple : par exemple, la fonction de génération d'un transistor

NMOS possède dix paramètres. Les techniques utilisées pour le développement de ces générateurs de dispositifs élémentaires ne seront pas développées dans ce manuscrit, car elles ont été introduites par M. Dessouky dans sa thèse[Des01] et sont développées dans la thèse de V. Bourguet. La figure 4.9 illustre la migration technologique d'un transistor PMOS : à gauche pour un procédé de fabrication de 0,6 μm, à droite pour un procédé de fabrication 0,13 μm.

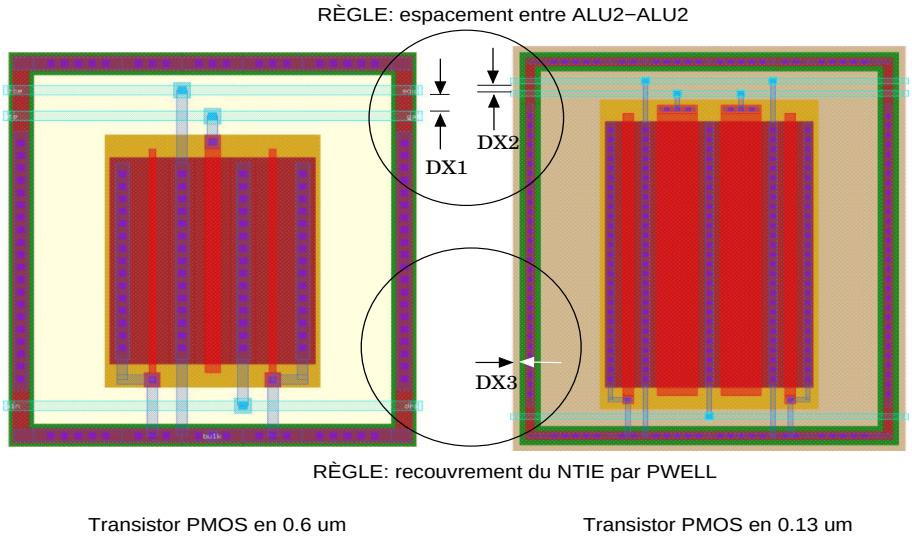


FIG. 4.9: Illustration de la migration technologique

- Enfin, les règles de dessin et les caractéristiques électriques du procédé de fabrication choisi interviennent dans le tracé fin des fils d'interconnexion entre les dispositifs élémentaires. Pour minimiser les risques de couplages capacitifs parasites, on a fait le choix de réaliser le routage dans des canaux de routage situés à l'extérieur des dispositifs élémentaires (et non par dessus les zones actives). Il est donc possible d'utiliser des techniques simples de routage symbolique, où la largeur des canaux d'interconnexion est calculée automatiquement en fonction de la technologie cible. Il faut simplement que la fonction d'évaluation des capacités parasites liées aux fils de routage soit paramétrée pour le procédé de fabrication choisi.

4.9 Conclusion

La méthodologie mise en place est le résultat d'une évolution progressive de la volonté de créer des générateurs de dessin des masques réutilisables avec CAIRO, vers la prise en compte du dimensionnement électrique avec CAIRO+. La formalisation de l'expertise du concepteur analogicien a été rendue possible par l'étude, en coopération étroite avec des concepteurs analogiciens, des méthodes de dimensionnement électrique des circuits analogiques.

Chapitre 5

Exploration du domaine de conception

Sommaire

5.1 Présentation générale 40
5.2 Étapes du DSES 41
5.3 Interface fonctionnelle 42
5.3.1 Procédures 43
5.3.2 Paramètres 44
5.4 Exploration du domaine de conception : DSES 46
5.4.1 Communication descendante 46
5.4.2 Communication ascendante 47
5.4.3 Mise en œuvre de l'exploration du domaine de conception . . . 48
5.5 Résumé 50

Le dimensionnement électrique et l'exploration du domaine de conception (ou DSES, **D**esign **S**pace **E**xploration and **S**izing) s'appuient sur l'expertise du concepteur analogicien. Ce chapitre détaille les mécanismes implantés dans le langage CAIRO+ pour aider le concepteur à formuler cette expertise.

5.1 Présentation générale

Le DSES se déroule en deux étapes : l'exploration à proprement parler et le dimensionnement électrique. L'exploration du domaine de conception (partie **DSE**) consiste à explorer l'espace multidimensionnel constitué par l'ensemble des valeurs que peuvent prendre les paramètres « électriques » des N dispositifs élémentaires instanciés (W et L pour les transistors MOS, valeurs R, L et C des capacités, inductances et résistances).

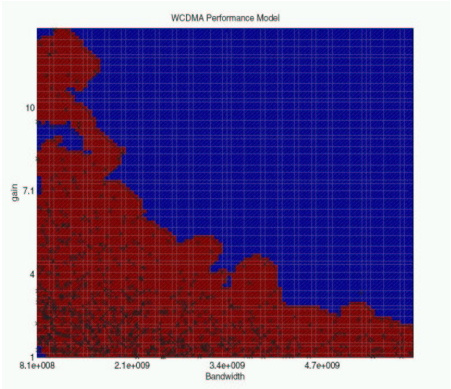


FIG. 5.1: Domaine de conception

La figure 5.1 tirée de [BV04] donne un exemple de domaine de conception où le gain est représenté en fonction du produit gain-bande (projection bidimensionnelle des modèles de performances P sur le plan produit gain-bande). La surface rouge — qui comporte des lacunes — est l'ensemble des solutions réalisables.

Dans l'approche CAIRO+, un générateur de module est destiné à être réutilisé par un autre générateur plus complexe. Par conséquent, un générateur ne se contente pas de générer un dessin des masques et un schéma non dimensionné, il a la capacité d'être interrogé quant à la faisabilité de spécifications électriques qui lui sont demandées par le niveau hiérarchique supérieur. Le générateur interrogé répond alors de deux manières : soit la génération est possible (pour le procédé de fabrication cible et pour les spécifications demandées), soit le procédé de fabrication ne permet pas de répondre aux spécifications. Dans ce second cas, le générateur demandeur en est informé. Ce jeu de questions-réponses a lieu à chaque niveau de la hiérarchie. Il permet au générateur de plus haut niveau la recherche d'un compromis. Le dimensionnement électrique définitif intervient après qu'une solution acceptable pour les niveaux hiérarchiques inférieurs a été déterminée.

Tout générateur possède une interface d'interrogation commune et standardisée. Chaque question est traitée par une procédure, laquelle peut rendre plusieurs réponses. Dans une approche basée sur la capitalisation de la connaissance, une ou plusieurs équations mo-

délisant le dispositif implanté permettent d'évaluer la réponse. Au niveau dispositif élémentaire, les équations sont issues du modèle électrique (BSIM3V3, BSIM4...). Elle font partie d'une bibliothèque mise à la disposition de l'utilisateur[1]. Au niveau module, il est de la responsabilité du concepteur de décrire correctement le comportement du module en implantant les services (équations) rendus adéquats.

5.2 Étapes du DSES

L'exploration du domaine de conception et le dimensionnement électrique consistent, à chaque niveau hiérarchique, à poser des « questions » aux modules fils de ce niveau, et à en retirer des « réponses » afin de trouver des valeurs de paramètres qui répondent aux spécifications. Le DSES comporte deux phases :

1. Une phase d'interrogation (l'exploration du domaine de conception) assurée par des procédures d'interrogation. Celles-ci concourent à dimensionner un niveau hiérarchique particulier, mais ne le dimensionnent pas complètement. On parlera de **procédures incomplètes**.
2. Une phase d'exécution (le dimensionnement électrique) assurée par des procédures d'exécution. On parlera de **procédures complètes**. Une procédure complète appartenant à un module M garantit que tous les dispositifs élémentaires instanciés par M ont été dimensionnés.

L'exploration du domaine de conception utilise la hiérarchie définie par le schéma non dimensionné, c'est à dire l'arbre des modules. À chaque niveau hiérarchique une procédure DSES attachée à un module M a en charge :

1. la propagation des spécifications électriques vers les modules fils, leur interrogation et la récupération de leurs réponses réalisables ;
2. la caractérisation des performances du module M consistant à fixer les valeurs d'un ou plusieurs paramètres électriques du module M ;
3. la transmission vers le niveau supérieur des cas d'échec.

Cette liste d'actions implique qu'une procédure de niveau N acceptant des contraintes de niveau (N+1) les traduise en contraintes moins abstraites pour le niveau N-1 afin de répondre à la question qui lui est posée. En effet, celle-ci est en mesure de poser des questions au niveau hiérarchique directement inférieur de manière à pouvoir former la réponse destinée au niveau supérieur. La figure 5.2 montre le parcours de l'arbre des modules lors de l'appel d'une procédure du DSES et la communication qui s'établit à ce moment. On peut constater qu'une seule question posée au niveau de la racine entraîne l'exploration complète en profondeur d'abord de l'arbre des modules, de la racine jusqu'aux feuilles. Plus exactement, pour N questions, il y a N+1 explorations :

- N explorations lors de la phase d'interrogation

[1] ce travail fait partie d'une autre thèse au laboratoire LIP6/ASIM

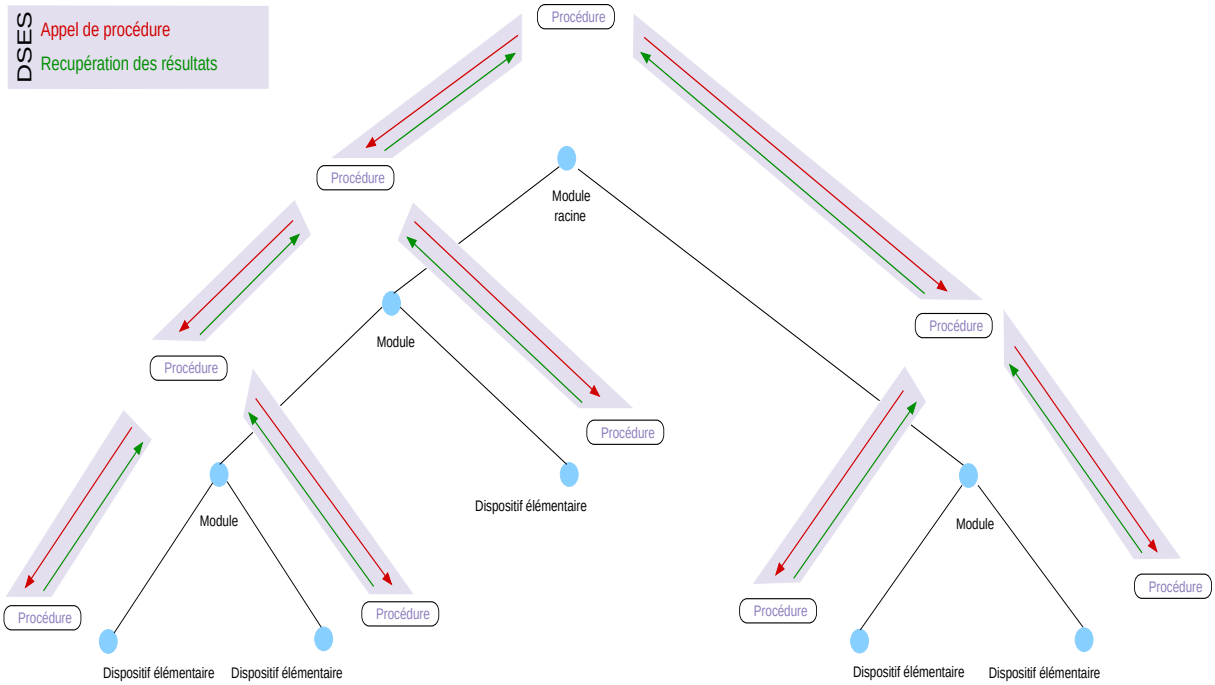


FIG. 5.2: Parcours de l'arbre des modules lors d'un DSES

- 1 exploration de la phase d'exécution

Le DSES commence avec une question posée au module racine. La procédure appelée appartenant à ce module récupère les paramètres électriques dont elle a besoin, puis les traduit pour le niveau inférieur. Elle positionne ensuite ces paramètres de plus bas niveau en entrée des procédures de ses fils, puis appelle leurs procédures (flèches rouges). Elle en retire enfin des informations (flèches vertes) qui lui servent à construire la réponse. Le processus est localement itératif (il y a une négociation entre deux niveaux successifs) et globalement récursif (la négociation est propagée jusqu'aux feuilles de l'arbre). À la fin, les dispositifs élémentaires sont interrogés.

Cependant, une dichotomie stricte entre les procédures incomplètes, qui ne permettent pas de dimensionner complètement le module auquel elles appartiennent, et les procédures complètes, qui dimensionnent entièrement, ne peut être établie. L'exploration du domaine de conception se déroule indifféremment avec l'un ou l'autre type de procédure suivant le choix de l'implantation fait par le concepteur analogicien, mais en définitive au moins une procédure complète de chaque niveau hiérarchique doit avoir été appelée si l'on veut dimensionner le système complet.

5.3 Interface fonctionnelle

L'interface fonctionnelle d'un module est une composante essentielle de tout générateur. Elle définit les procédures de calcul qui correspondent aux questions qu'il est possible

de poser et aux réponses que l'on peut attendre.

Les relations entre les éléments de l'interface fonctionnelle sont illustrées par la figure 5.3. Le générateur possède un jeu de procédures (les questions) et de paramètres liés aux procédures (les variables des questions). Dans une approche basée sur la représentation des connaissances, une procédure repose sur un ensemble d'équations modélisant le comportement du module, et permettant de calculer les valeurs de certains paramètres à partir de valeurs fixées pour certains autres.

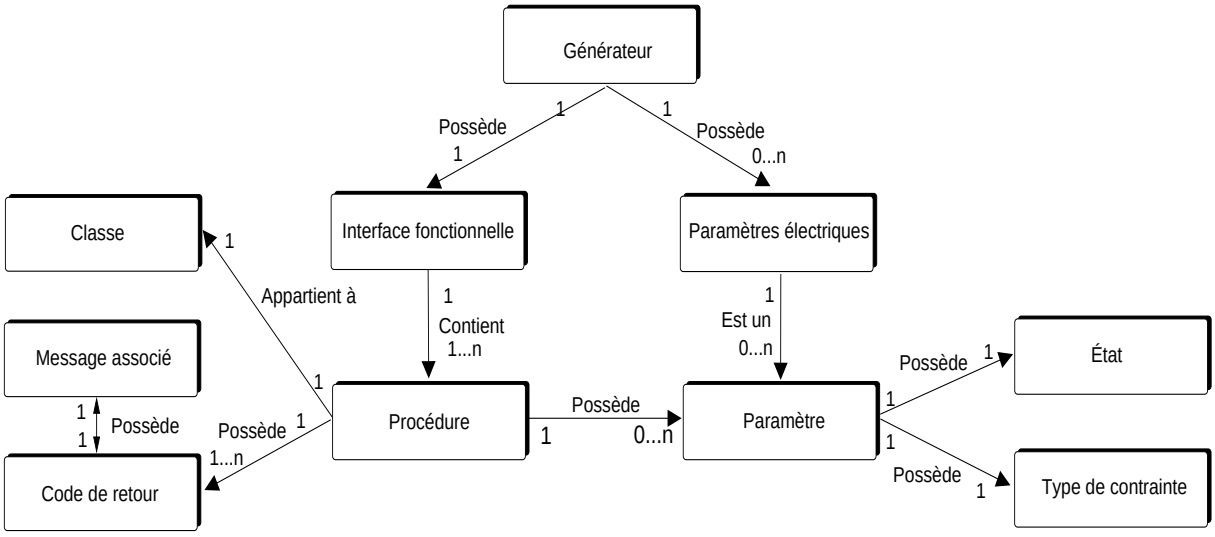


FIG. 5.3: L'interface fonctionnelle

Une procédure possède une classe (permettant de savoir si elle dimensionne complètement le module) et un ensemble de codes de retour définissant exhaustivement tous les cas d'échecs. À tout instant, un paramètre possède un *état* qui indique si sa valeur est définie ou non. Enfin, pour chaque procédure, un paramètre possède un type qui dépend des contraintes qui lui sont associées.

5.3.1 Procédures

L'exploration du domaine de conception est menée à bien par un ensemble de procédures de dimensionnement. Un module ou un dispositif élémentaire doit comporter au moins une procédure, mais en fournit généralement plusieurs. Toute procédure est un service rendu au niveau supérieur (figure 5.4).

Classe de procédure

CAIRO+ définit deux classes de procédure :

- Une procédure complète est une procédure qui, lorsqu'elle est appelée, permet d'entièrement dimensionner électriquement tous les dispositifs élémentaires du module auquel elle appartient.

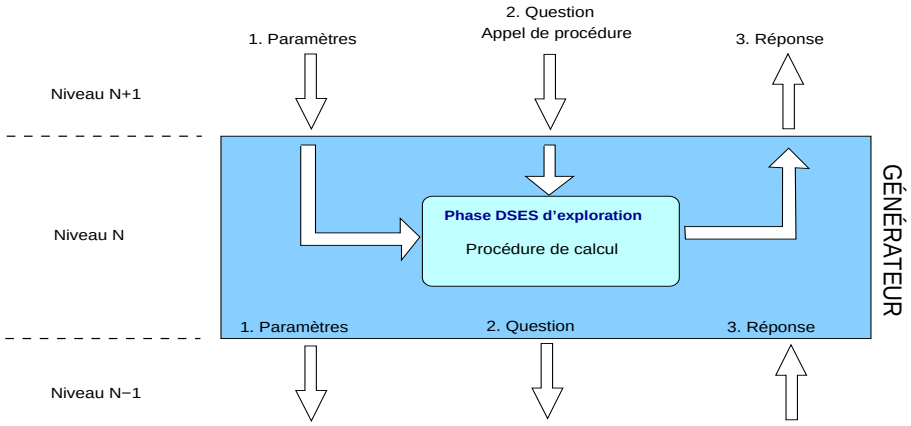


FIG. 5.4: Principe de fonctionnement d'une procédure

- Une procédure incomplète ne garantit pas le dimensionnement complet de tous les dispositifs élémentaires du module auquel elle appartient.

La classe des procédures est notée

```
CP_COMPLETE   pour une procédure complète
CP_UNCOMPLETE pour une procédure incomplète
```

Paramètres attachés

Une procédure possède un ensemble de paramètres attachés (ce sont les paramètres d'entrée-sortie de la procédure). Une procédure peut accepter plusieurs paramètres d'entrée et plusieurs paramètres de sortie (elle a la possibilité de répondre à plusieurs questions).

Code de retour d'une procédure

Une procédure possède un ensemble de codes de retour définis par le concepteur de la procédure. Ces codes de retour permettent de lister explicitement tous les cas possibles d'échec de la procédure, afin de faciliter la mise en place de l'exploration du domaine de conception. Un message est associé à chaque code de retour ; il est ainsi possible de tracer une suite d'appels à des procédures de dimensionnement identifiées, facilitant la mise au point des générateurs.

5.3.2 Paramètres

À chaque module M de niveau N sont associés un certain nombre de paramètres électriques qui caractérisent le comportement électrique du module. Ces paramètres sont connus des procédures des modules de niveau (N+1) qui peuvent appeler une procédure de niveau N, en imposant des contraintes sur certains paramètres du module M.

Toute procédure de dimensionnement du module M consiste donc à calculer les valeurs des paramètres libres de M en utilisant les valeurs des paramètres *fixés* qui sont définis par la fonction appelante.

Un même paramètre peut intervenir dans plusieurs procédures. On dit qu'un paramètre est attaché à une procédure si les équations mises en jeu dans la procédure font intervenir la valeur de ce paramètre.

Type

Il est à noter que le type d'un paramètre n'est pas une propriété intrinsèque, mais dépend de la procédure à laquelle est attaché le paramètre.

- On appelle paramètre d'entrée de la procédure P un paramètre dont la valeur est imposée par la procédure appelante ;
- On appelle paramètre de sortie de la procédure P un paramètre dont la valeur est calculée par la procédure P ;
- Certains paramètres peuvent n'être que partiellement contraints par la procédure appelante. Celle-ci peut définir un intervalle de variation autorisé. Dans ce cas, la procédure appelée doit calculer la valeur finale du paramètre en respectant la contrainte imposée. De tels paramètres qui sont à la fois soumis à une contrainte par la fonction appelante et calculés par la fonction appelée sont appelés paramètres d'entrée-sortie de la procédure P.

On a donc cinq types selon la contrainte appliquée :

TAB. 5.1: Type d'un paramètre

Type	Contrainte
CP_RANGE	intervalle (valeur1 $\leq$ X $\leq$ valeur2)
CP_INF	borne inférieure large (X $\geq$ valeur)
CP_SUP	borne supérieure large (X $\leq$ valeur)
CP_FIXED	égalité (X = valeur)
CP_NO	pas de contrainte

Le langage CAIRO+ fournit au concepteur d'une fonction de dimensionnement les primitives permettant d'imposer une valeur fixée sur les paramètres de type entrée, ou d'imposer un intervalle de variation sur les paramètres en entrée-sortie. Il fournit au concepteur les moyens de récupérer les résultats calculés par la fonction appelée.

État d'un paramètre

À tout paramètre d'un module M est associée une variable d'état DEFINED qui permet de gérer la « durée de vie » des valeurs calculées pour les paramètres électriques du module M.

- Initialement, tous les paramètres d'un module ont une valeur indéfinie.
- Un paramètre passe dans l'état défini soit parce qu'il est défini comme paramètre d'entrée (par la procédure appelante), soit parce que sa valeur est calculée par la procédure appelée.

Cependant, la « durée de vie » de la valeur d'un paramètre n'est pas infinie. En effet, il y a généralement plusieurs procédures de dimensionnement associées à un même module M ; les différentes procédures, correspondant à des contraintes différentes sur les paramètres d'entrée, ne calculent pas nécessairement les valeurs de tous les paramètres libres du module M. Au cours de deux appels successifs à deux procédures **P** et **P'** d'un même module M, la valeur du paramètre **A** calculée par **P** peut être rendue invalide par l'exécution de la procédure **P'**. Par conséquent, CAIRO+ fournit la garantie suivante : à la fin de l'exécution d'une procédure **P** seuls les paramètres attachés à la procédure (et déclarés dans le prototype de la procédure) seront dans un état défini. Tous les autres paramètres du module passent dans l'état indéfini. Cette variable d'état gérée automatiquement par le langage CAIRO+ permet d'implanter des vérifications chaque fois qu'une procédure cherche à accéder à la valeur d'un paramètre.

5.4 Exploration du domaine de conception : DSES

L'exploration du domaine de conception est réalisée par un jeu de questions-réponses entre les niveaux hiérarchiques, chaque question qu'il est possible de poser représentant un service rendu, la réponse devant concourir à déterminer un dimensionnement électrique valide par exploration (restreinte) des différentes possibilités offertes suivant les conditions initiales imposées. Un niveau hiérarchique (un module) a la possibilité d'interroger chacun de ses fils directs (communication descendante) tandis qu'il peut être interrogé par son père direct (communication ascendante).

5.4.1 Communication descendante

Une procédure rendant un service est destinée à être appelée du niveau supérieur. Elle a besoin de paramètres en entrée, paramètres qui doivent être positionnés par l'appelant. Du côté de la procédure appelée, il faut récupérer les paramètres. L'enchaînement de ces actions constitue la partie descendante de la communication entre les générateurs des différents niveaux hiérarchiques (figure 5.5).

Le concepteur d'un générateur de niveau (N+1) utilisateur d'une procédure appartenant à un générateur de niveau N doit positionner ses paramètres d'entrée avant de l'appeler (figure 5.5, point 1). Le nombre de paramètres fixés varie suivant la procédure.

Les contraintes sur les paramètres d'entrée sont mémorisées au niveau du générateur N en attendant d'être utilisées. La mémorisation est persistante : la valeur du paramètre ne change pas tant que celle-ci n'a pas été explicitement modifiée.

La récupération des données est faite par le générateur de niveau N lorsque la procédure est appelée.

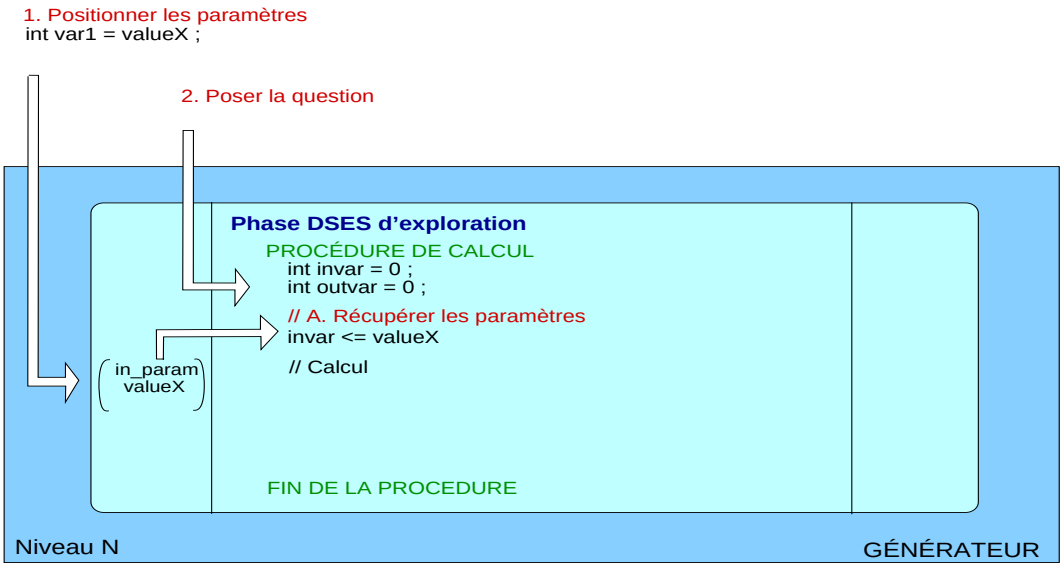


FIG. 5.5: Communication descendante

Toutes les valeurs des paramètres dont a besoin la procédure sont ainsi récupérées, ce qui lui permet de réaliser le calcul (service) associé (figure figure 5.5, point A). Elle peut éventuellement interroger le niveau hiérarchique inférieur de la même manière.

5.4.2 Communication ascendante

La procédure ayant réalisé le calcul doit retourner les informations requises par le niveau supérieur. Il s'agit de paramètres de sortie qui sont ensuite récupérés par le demandeur (figure 5.6).

La communication ascendante se déroule en deux phases. La première a lieu lorsque la procédure de dimensionnement est appelée ; elle consiste à réaliser le calcul demandé, puis à mémoriser le résultat pour chaque paramètre de sortie. La procédure appelée renvoie aussi explicitement un code de retour correspondant à son état, ce qui permet de déterminer de manière non ambiguë la raison d'un éventuel échec.

La réponse qu'un générateur est susceptible de donner est donc l'ensemble du code de retour de la procédure appelée et des valeurs des paramètres de sortie.

La seconde phase a lieu après l'appel de la procédure de dimensionnement lorsque le générateur de niveau (N+1) récupère les paramètres de sortie de la procédure qu'il vient d'appeler (figure 5.6, point 3). La réponse donnée est formée de la valeur de chaque paramètre et du code de retour de la procédure.

Si la procédure appelée ne s'est pas correctement déroulée (une spécification non réalisable par exemple), la procédure appelante doit déterminer les actions à effectuer en fonction de la valeur du code de retour de la procédure.

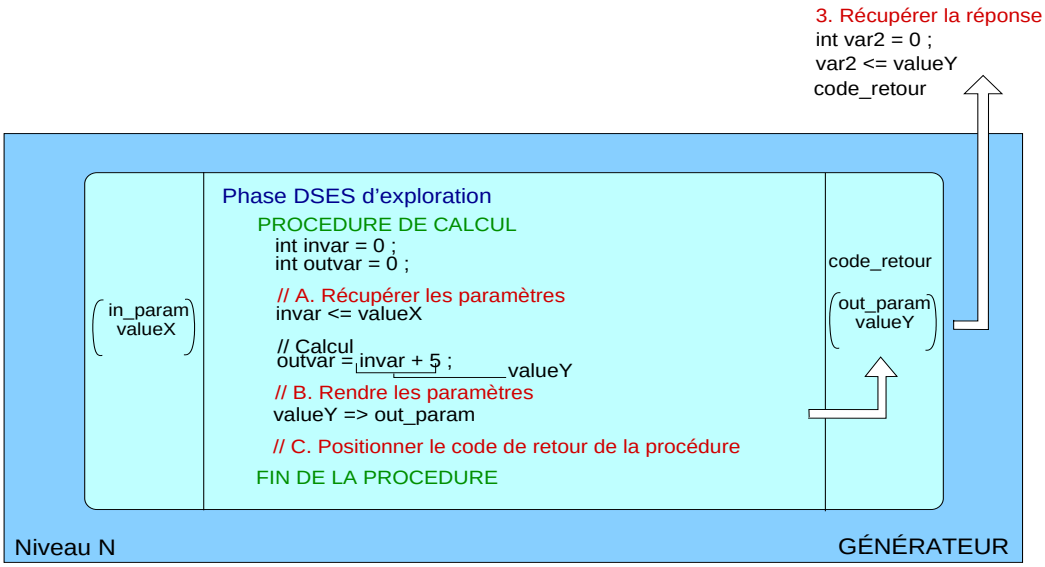


FIG. 5.6: Communication ascendante

5.4.3 Mise en œuvre de l'exploration du domaine de conception

Une procédure possède un code de retour qui doit être systématiquement testé par la procédure appelante : CAIRO+ ne propose pas d'outils réalisant automatiquement l'optimisation du dimensionnement électrique. La stratégie d'optimisation est entièrement à la charge du concepteur, dont la tâche est tout de même facilitée par le cadre proposé par CAIRO+, cadre permettant de formaliser son raisonnement.

Appel d'une procédure, examen et traitement des échecs

La procédure d'optimisation du dimensionnement électrique, à la charge du concepteur, doit nécessairement comporter la séquence d'action suivante :

```
// 1. Positionner les paramètres d'entrée de la procédure
// 2. Appel à la procédure de dimensionnement

SI code_procédure != VALID
  // 3. Examen du code de retour de la procédure
  // 4. Modifier les paramètres d'entrée en conséquence
FIN SI
```

Les paramètres doivent être positionnés avant l'appel effectif à la procédure de dimensionnement. Si le code de retour de la procédure est différent de la réussite, il doit être examiné pour déterminer la raison de l'échec (le code de retour en indique de façon spécifique et unique la raison). L'ensemble des cas peut être traité par le concepteur du générateur à l'aide d'une structure du type :

```
SUIVANT LE CODE DE RETOUR
  CAS 1: // Action 1 : modification des paramètres de la procédure appelée
  CAS 2: // Action 2 : modification des paramètres de la procédure appelée
  CAS 3: // Action 3 : modification des paramètres de la procédure appelée
      ...
FIN
```

Notons que la modification des paramètres relève de l'expertise du concepteur. Il lui appartient d'inclure cette séquence dans une boucle ou dans toute autre structure de contrôle qu'il jugera bon d'utiliser. La documentation du générateur qu'il écrit, et surtout des codes d'échecs associés à chaque procédure de dimensionnement, est donc indispensable, de sa qualité dépend son utilisation future.

Si un cas d'échec ne peut être résolu, le concepteur a la possibilité de la prendre en compte de manière létale au niveau courant (erreur fatale) ou la signaler au niveau supérieur, qui a alors le loisir de la traiter, de générer une erreur fatale ou de reporter le problème au niveau supérieur (figure 5.7).

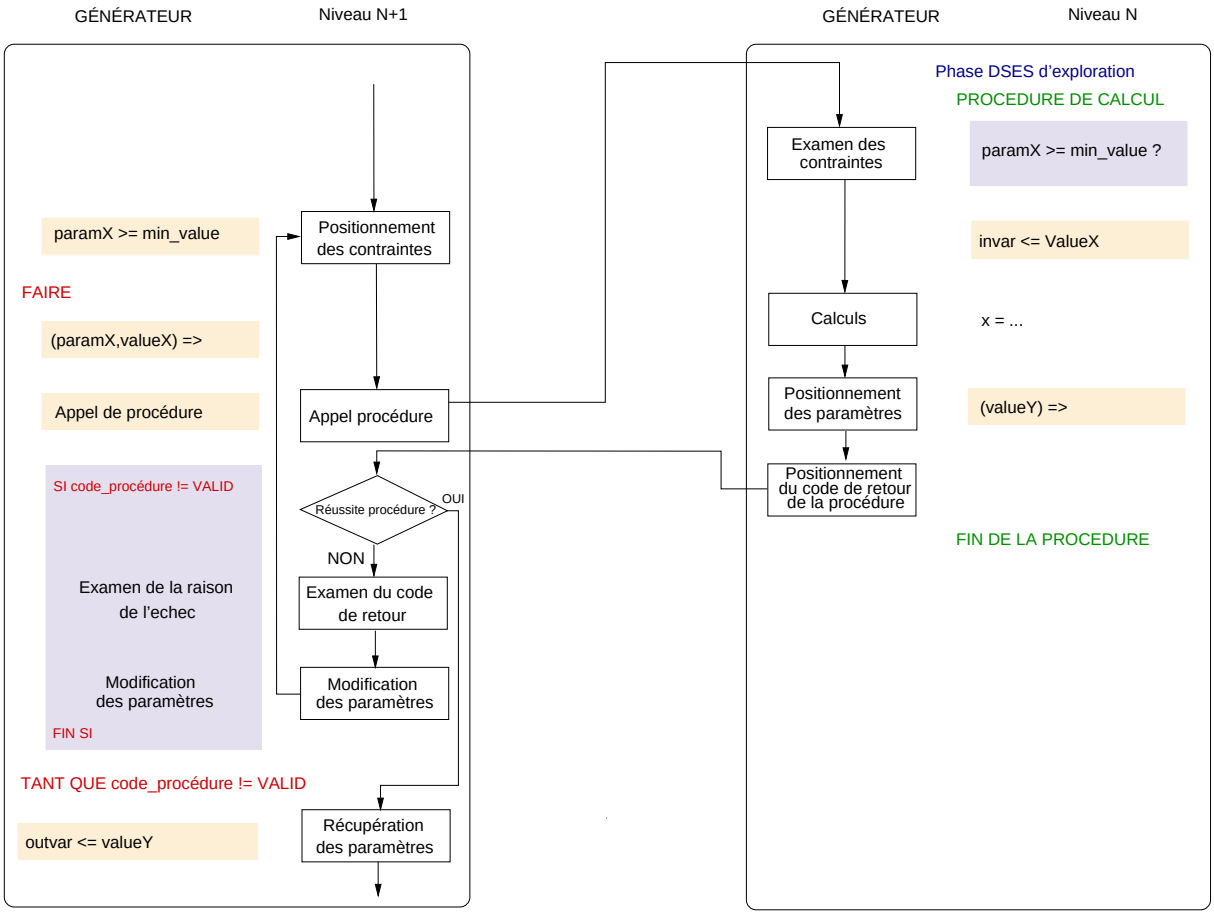


FIG. 5.7: Organigramme d'un DSES et les blocs de contrôle

Le problème peut ainsi être propagé vers la racine du circuit, ce qui indique en général une impossibilité de réaliser les performances demandées avec les spécifications imposées. Une solution consiste alors à changer les spécifications au niveau du programme principal, ou à changer le procédé de fabrication.

5.5 Résumé

Ce chapitre a présenté la méthodologie utilisée dans la phase de dimensionnement électrique. On en retiendra que les générateurs de CAIRO+ sont des objets intelligents capables de communiquer entre eux. Le concepteur analogicien est guidé dans sa formalisation et dans son abstraction des procédures hiérarchiques de dimensionnement. Ce faisant, ses connaissances sont capitalisées. La réussite du processus repose sur :

- une décomposition hiérarchique du circuit et une communication entre niveaux hiérarchiques à travers des services rendus ;
- l'absence de simulation électrique, rejetée à la fin du cycle de conception ;
- l'indépendance vis à vis du procédé de fabrication, puisque celui-ci n'intervient qu'à travers l'interface fonctionnelle des dispositifs élémentaires ;
- le souci de la réutilisation : le concepteur d'un module est incité à écrire les procédures de dimensionnement qui permettront d'instancier ce module dans d'autres modules.

Chapitre 6

Placement optimisé d'objets déformables

Sommaire

6.1 Analyse du problème 52
6.2 Fonction de forme d'un dispositif élémentaire 53
6.3 Fonctions de Forme sous contrainte FFDX et FFDY 54
6.4 Construction ascendante de la Fonction de Forme d'un module 55
6.4.1 Fonctions de forme sous contrainte d'un conteneur vertical . . . 55
6.4.2 Fonctions de forme sous contrainte d'un conteneur horizontal . 56
6.4.3 Fonction de forme d'un module 56
6.5 Propagation descendante des contraintes de forme 57
6.6 Implantation 57
6.6.1 Structure de données 57
6.6.2 Algorithme de composition pour un conteneur horizontal 59
6.6.3 Exemple 60
6.7 Conclusion 61

À la fin de l'étape de dimensionnement électrique d'un module M (phase DSES), tous les dispositifs élémentaires instanciés dans l'arbre de modules dont M est la racine ont été dimensionnés du point de vue « électrique », c'est à dire que les valeurs des paramètres des dispositifs élémentaires sont figées (paramètres W et L pour les transistors, paramètres R,L,C pour les résistances, inductances ou capacités). L'étape suivante est la génération effective du dessin des masques de chacun de ces dispositifs élémentaires et le placement optimisé afin de préparer l'étape de routage.

6.1 Analyse du problème

La difficulté du problème tient au caractère générique des modules décrits avec le langage CAIRO+ : pour un même schéma de principe, différentes spécifications fonctionnelles vont aboutir à des dimensions géométriques très différentes pour les dispositifs élémentaires instanciés. De plus, entre deux réalisations d'un même module, les surfaces des différents dispositifs élémentaires ne varient pas nécessairement dans le même sens : certains vont grossir, alors que d'autres vont voir leur surface diminuer. Enfin, l'objectif de portabilité technologique augmente la difficulté du problème : les valeurs des capacités ne varient pas de la même façon que le courant des transistors quand on passe d'un procédé de fabrication 0,25 μm à un procédé 0,12 μm.

La difficulté à résoudre est donc de placer dans une forme rectangulaire un ensemble d'objets (les dispositifs élémentaires) alors que les surfaces sont extrêmement variables et que l'encombrement effectif de chacun des dispositifs élémentaires n'est connu qu'à la fin de l'étape de dimensionnement DSES.

Heureusement, on dispose d'un degré de liberté important : les dispositifs élémentaires ont des surfaces variables, et a priori quelconques, mais ce sont des objets déformables. En effet, pour un dimensionnement électrique défini, il existe plusieurs réalisations physiques équivalentes : par exemple, pour minimiser les capacités de drain, les transistors « analogiques » ont généralement une structure « en peigne » où un seul transistor fonctionnel est réalisé par la mise en parallèle de plusieurs transistors élémentaires (figure 6.1). En faisant varier le nombre de transistors élémentaires on peut donc faire varier le facteur de forme (c'est-à-dire le rapport hauteur sur largeur DY/DX). De même, les capacités analogiques sont généralement réalisées comme des matrices de capacités élémentaires car il est plus facile de contrôler précisément les valeurs relatives de plusieurs capacités que de contrôler les valeurs absolues. En faisant varier le nombre de colonnes de cette matrice de capacités élémentaires (sans modifier le nombre de capacités élémentaires), on peut donc

faire varier le facteur de forme sans modifier la valeur de la capacité.

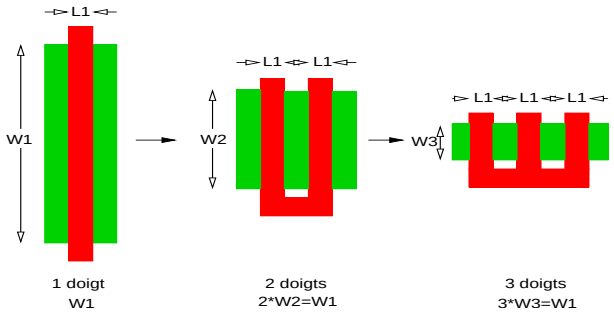


FIG. 6.1: Illustration du repliement d'un transistor

Il est remarquable que, pour un dimensionnement électrique donné, les différentes réalisations physiques d'un même dispositif élémentaire aient des surfaces sensiblement égales. Nous décrivons dans ce chapitre comment exploiter cette propriété pour générer un placement optimisé.

6.2 Fonction de forme d'un dispositif élémentaire

On appelle fonction de forme d'un dispositif élémentaire la fonction qui définit les différentes réalisations physiques { DE(n) } d'un dispositif élémentaire, lorsqu'on a fixé ses paramètres électriques. Toutes les réalisations physiques DE(n) sont a priori équivalentes du point de vue électrique (si on ne tient pas compte des effets parasites). La fonction de forme est donc une fonction discrète qui dépend généralement d'une unique variable entière n. Pour un transistor en peigne, n correspond au nombre de transistors élémentaires. Pour une matrice de capacité, n correspond au nombre de colonnes. Il s'agit donc d'une fonction de l'ensemble $\mathbb{N}^+$ vers l'ensemble { DE(n) } des réalisations physiques du dispositif élémentaire.

En pratique le nombre des réalisations physiques possibles est fini :

- Dans le cas d'un transistor en peigne, et pour une largeur de canal totale W, la valeur de n est bornée par nmax = W / W_{min} où W_{min} est la largeur de canal minimale d'un transistor élémentaire. La valeur W_{min} est imposée par le procédé de fabrication choisi.
- Pour une matrice de capacités, et pour une valeur totale de la capacité C, la valeur de n est bornée par nmax = C / C_{unit} où C_{unit} est la valeur de la capacité unitaire, car il ne peut pas y avoir plus de colonnes que de capacités unitaires. Ici encore, la valeur de C_{unit} est définie par le procédé de fabrication choisi.

Pour un dispositif élémentaire donné, il existe donc une fonction de forme spécifique pour chaque dimensionnement électrique de ce dispositif, et cette fonction est une fonction de l'ensemble des entiers compris entre 1 et nmax vers l'ensemble des réalisations physiques { DE(n) } (figure 6.2).

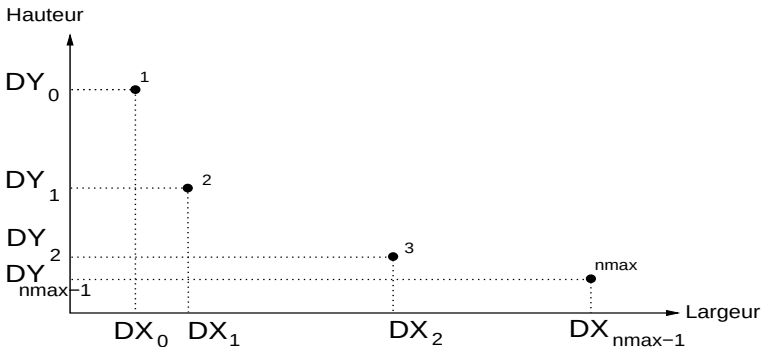


FIG. 6.2: Fonction de forme

6.3 Fonctions de Forme sous contrainte FFDX et FFDY

Soit un dispositif élémentaire dimensionné et l'ensemble de ses réalisations physiques { DE(n) }. Une réalisation physique particulière DE(n) possède une largeur DX_n et une hauteur DY_n. La fonction de forme définit donc un ensemble fini de couples (DX_n, DY_n), et on peut ainsi définir une bijection entre les deux ensembles DX_n et DY_n, permettant d'obtenir la hauteur DY_n connaissant la largeur DX_n et réciproquement.

À partir de cette bijection, il est possible de définir quatre nouvelles fonctions que nous appellerons fonctions de forme sous contrainte :

- Les deux fonctions DXFFX(xmax) et DYFFX(xmax) donnent la largeur effective DX et la hauteur effective DY du dispositif élémentaire quand on impose contrainte de largeur maximale xmax.
- Les deux fonctions DXFFY(ymax) et DYFFY(ymax) donnent la largeur effective DX et la hauteur effective DY du dispositif élémentaire quand on impose contrainte de hauteur maximale ymax.

Ces quatre fonctions sont des fonctions associant un élément de l'ensemble des réels positifs $\mathbb{R}^+$ vers l'ensemble discret des largeurs possibles DX_n ou vers l'ensemble discret des hauteurs possibles DY_n (figures 6.3 et 6.4).

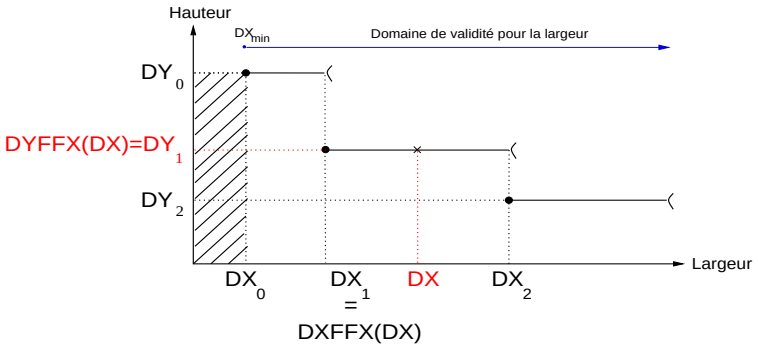


FIG. 6.3: Fonctions de forme DXFFX et DYFFX sous la contrainte DX

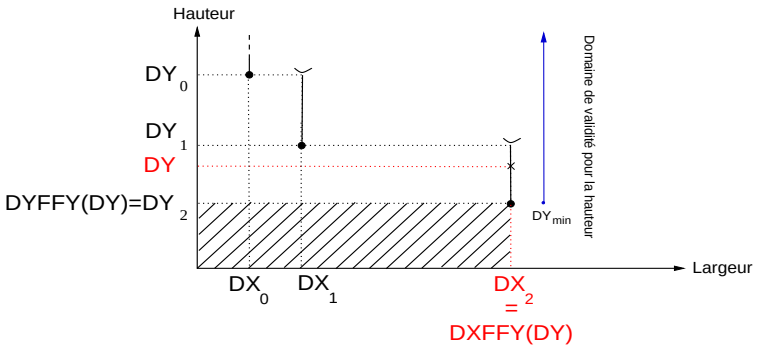


FIG. 6.4: Fonctions de forme sous la contrainte DY

Il est important de noter que les deux fonctions de forme sous contrainte DYFFX et DXFFY sont monotones décroissantes. La surface occupée par les différentes réalisations DE(n) étant approximativement constante, toutes ces courbes ont en effet la forme générale d'un arc d'hyperbole.

6.4 Construction ascendante de la Fonction de Forme d'un module

Lorsqu'un module M est entièrement dimensionné (c'est à dire lorsque les paramètres de tous les dispositifs élémentaires qu'il contient ont des valeurs fixées), il est possible de définir une fonction de forme pour le module M. Plus précisément, nous allons définir un mécanisme général de composition des fonctions de forme permettant de calculer la fonction de forme d'un module M à partir des fonctions de forme des modules qu'il contient, pourvu qu'on ait défini le placement relatif en utilisant la technique des conteneurs.

Conformément au modèle des « slicing tree », la topologie d'un module est obtenue en aboutant alternativement des conteneurs horizontaux et verticaux.

6.4.1 Fonctions de forme sous contrainte d'un conteneur vertical

Soit un conteneur vertical V obtenu par empilement vertical de conteneurs horizontaux H_i et soumis à une contrainte de largeur maximale xmax. On suppose qu'on connaît les fonctions de forme sous contrainte de chacun des conteneurs H_i : la fonction $DXFFX_{H_i}(xmax)$ donne la largeur effective DX du conteneur H_i quand on lui impose une largeur maximale xmax. La fonction $DYFFX_{H_i}(xmax)$ donne la hauteur effective DY du conteneur H_i quand on lui impose une largeur maximale xmax.

Il est clair que la hauteur du conteneur vertical V soumis à une contrainte de largeur maximale est la somme des hauteurs des conteneurs empilés H_i soumis à la même contrainte. Par ailleurs, la largeur du conteneur vertical V soumis à une contrainte de largeur maximale est évidemment la valeur maximale des largeurs des conteneurs empilés. Il

est alors possible de définir les fonctions de forme sous contrainte du conteneur englobant V de la façon suivante :

$$\begin{aligned} DXFFX_V(xmax) &= \max_i\{DXFFX_{H_i}(xmax)\} \\ DYFFX_V(xmax) &= \sum_i DYFFX_{H_i}(xmax) \end{aligned} \tag{6.1}$$

Étant donné le mécanisme de construction présenté ci-dessus, la fonction $DYFFX_V$ est également monotone décroissante.

6.4.2 Fonctions de forme sous contrainte d'un conteneur horizontal

Pour un conteneur horizontal H obtenu par aboutement de conteneurs verticaux V_i et soumis à une contrainte de hauteur maximale ymax, on peut écrire de façon symétrique :

$$\begin{aligned} DXFFY_H(ymax) &= \sum_i DXFFY_{V_i}(ymax) \\ DYFFY_H(ymax) &= \max_i\{DYFFY_{V_i}(ymax)\} \end{aligned} \tag{6.2}$$

Pour les mêmes raisons que précédemment, la fonction $DXFFY_V$ est également monotone décroissante.

6.4.3 Fonction de forme d'un module

Un module M de premier niveau ne contient que des dispositifs élémentaires. À l'issue de la phase DSES, tous les dispositifs élémentaires ont été dimensionnés du point de vue électrique, et il est donc possible de construire la fonction de forme pour chacun des dispositifs élémentaires. En utilisant 6.1 et 6.2, il est possible de construire progressivement les fonctions de forme des différents conteneurs pour aboutir à la fonction de forme du module M. Il suffit, pour un conteneur vertical, de faire varier continûment la contrainte xmax entre zéro et l'infini, et d'en déduire l'ensemble des valeurs discrètes (DX_i,DY_i) au moyen des fonctions DXFFX et DYFFX définies en 6.1. On obtient ainsi l'ensemble exhaustif des réalisations possibles du conteneur minimisant la surface totale occupée par le conteneur. Pour un conteneur horizontal, il faut faire varier la contrainte ymax et utiliser les fonctions DXFFY et DYFFY définies en 6.2.

Il faut noter que l'ensemble des réalisations { MOD(n) } du module M construites par la procédure récursive présentée ci-dessus ne représente pas l'ensemble de toutes les réalisations physiques possibles du module M pour ce dimensionnement électrique. Le nombre total de réalisations possibles d'un module dimensionné est beaucoup plus grand : si on choisit arbitrairement une forme DE(i) pour chacun des dispositifs élémentaires dimensionnés constituant le module et qu'on aboute ces dispositifs élémentaires en respectant

la procédure d'aboutement définie par l'arborescence des conteneurs, on obtient une réalisation possible du module mais, en général, cette solution est sous-optimale du point de vue de la surface totale résultante. L'ensemble { MOD(n) } correspondant à la fonction de forme construite ci-dessus définit un ensemble de solutions optimales au sens de Pareto : pour une hauteur (ou une largeur) imposée, on obtient la réalisation physique possédant la surface minimale.

Remarquons enfin que chaque élément de l'ensemble { MOD(n) }, c'est-à-dire chaque point de la fonction de forme, a été obtenu en choisissant un point particulier de la fonction de forme de chacun des dispositifs élémentaires instanciés (c'est-à-dire une réalisation particulière de chacun des dispositifs élémentaires).

Il est donc possible — et nécessaire — de conserver pour chaque point de la fonction de forme du module M (c'est-à-dire pour chaque réalisation possible de ce module) l'ensemble des points correspondants des fonctions de forme des dispositifs élémentaires et modules qu'il contient, et qui ont permis de réaliser cette forme particulière du module M.

6.5 Propagation descendante des contraintes de forme

Nous avons vu dans le paragraphe précédent qu'une fois le dimensionnement électrique d'une arborescence de modules effectué il est possible de calculer, pour chaque module instancié dans cette arborescence, une fonction de forme { MOD(n) } qui définit l'ensemble des réalisations optimales de ce module du point de vue de la surface occupée. Cette construction des fonctions de forme est « ascendante », en ce sens qu'elle s'effectue au moyen d'un parcours récursif de l'arborescence des modules (ou plus exactement de l'arborescence des conteneurs) en remontant des feuilles (les dispositifs élémentaires) vers la racine de l'arborescence.

Le simple fait d'imposer une contrainte de forme sur le module racine de l'arbre des modules (c'est-à-dire une hauteur maximale ymax, ou une largeur maximale xmax pour le module de niveau hiérarchique le plus élevé) définit de façon unique la réalisation matérielle de chacun des modules de l'arborescence : en effet, imposer cette contrainte revient à choisir un point particulier de la fonction de forme (monotone décroissante) du module racine. Puisque — par construction — tout point de la fonction de forme d'un module définit de façon unique les formes des modules ou dispositifs élémentaires instanciés, cette contrainte se propage de façon descendante dans l'arbre des modules, jusqu'aux dispositifs élémentaires.

6.6 Implantation

6.6.1 Structure de données

La fonction de forme d'un dispositif élémentaire ou d'un module peut comporter plusieurs milliers de points suivant la manière de concevoir le dessin des masques et les dimensions physiques intrinsèques. Il faut donc une manière appropriée de représenter les

données sachant que la construction et la composition de fonctions de formes demandent de nombreuses insertions et suppressions. L'utilisation d'arbres AVL[AVL62] résout les problèmes de rapidité d'accès et de manipulation.

Une fonction de forme comporte les éléments suivants (figure 6.5) :

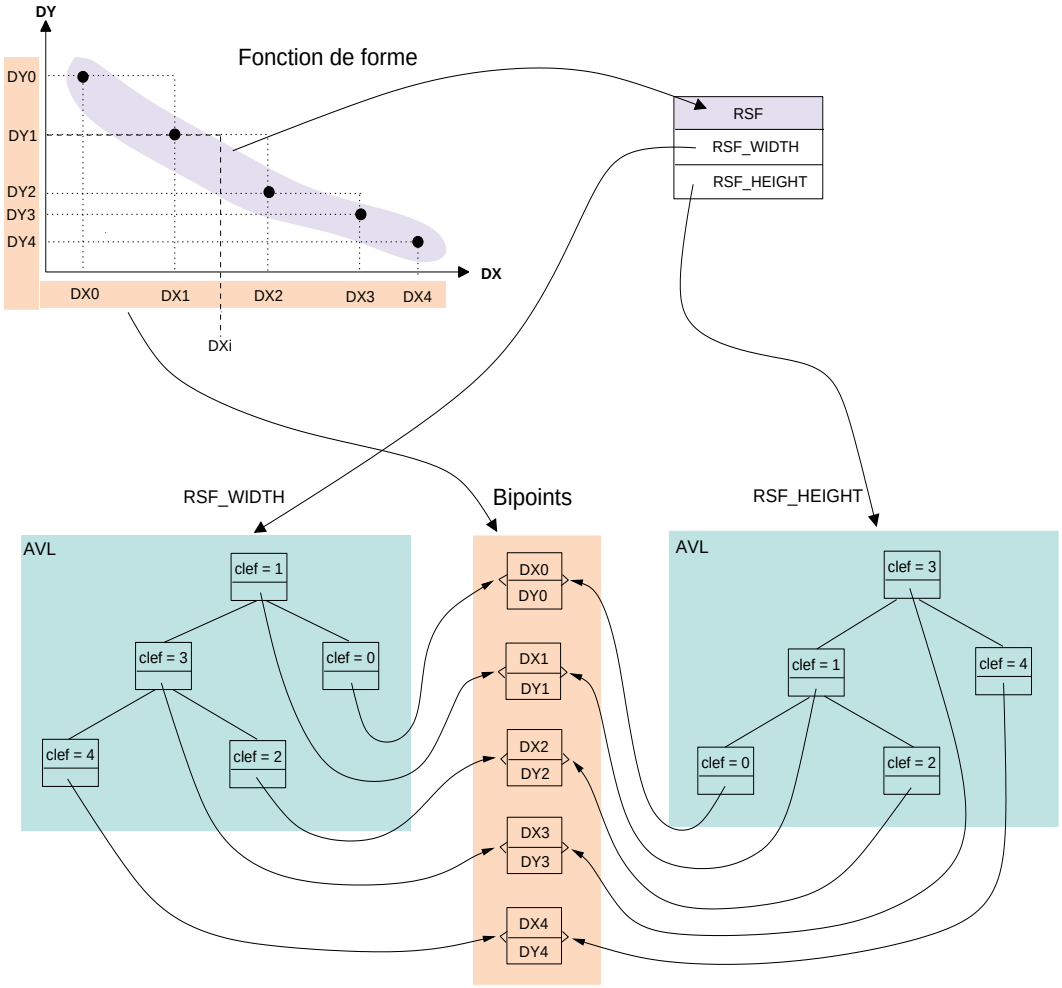


FIG. 6.5: Représentation en mémoire d'une fonction de forme

- Chaque point est représenté par ses coordonnées DX et DY (couple largeur-hauteur). La structure est nommée **bipoint**.
- Une fonction de forme peut se lire indifféremment en prenant l'abscisse ou l'ordonnée comme contrainte (lecture symétrique) pour trouver la coordonnée complémentaire, mais un AVL ne présente pas cette double entrée. Pour des raisons d'efficacité, on choisit de représenter la fonction de forme par un ensemble de deux AVL, l'un pour la hauteur (la contrainte est la hauteur et donne la largeur correspondante), l'autre pour la largeur (la contrainte est la largeur et donne la hauteur correspondante). Ces deux AVL se partagent les bipoints.

- La clef de recherche dans les AVL est donc soit la hauteur, soit la largeur.
- La donnée d'un noeud d'un AVL pointe sur le bipoint dont l'abscisse ou l'ordonnée forme la clef. Pour un AVL représentant la hauteur en fonction de la largeur, la clef est la hauteur. Inversement, un AVL représentant la largeur voit les données des noeuds pointer sur les bipoints dont la largeur est la clef d'entrée.

Les opérations menées sur une fonction de forme (et par là même, sur les AVL) sont :

- l'insertion d'un bipoint ;
- la suppression d'un bipoint ;
- la recherche d'un bipoint (la contrainte de largeur DX donne la hauteur réalisable DY, ou la contrainte de hauteur DY donne la largeur DX réalisable).

La suppression d'un bipoint se résume en la suppression d'une clef dans un AVL et ne sera pas abordée. En revanche, l'ajout d'un bipoint est plus complexe.

6.6.2 Algorithme de composition pour un conteneur horizontal

Une fonction de forme est représentée par deux AVL où ne sont mémorisés que les points de la fonction de forme, plus exactement les coordonnées DX et DY de chaque bipoint. Les points sont partagés entre les deux AVL, le premier étant utilisé pour une recherche suivant une contrainte de hauteur (la clef de recherche est une hauteur), le second pour une recherche suivant une contrainte de largeur (la clef de recherche est une largeur).

On suppose que la fonction de forme de chaque dispositif élémentaire existe déjà. Dans le cas d'une composition pour un conteneur horizontal, seul l'AVL des conteneurs fils permettant une recherche sur une contrainte de hauteur est utilisé. L'algorithme réalisant la composition pour un conteneur horizontal est récursif afin de remonter des feuilles vers la racine. Avant d'appliquer les équations 6.1 et 6.2, il faut s'assurer de l'intervalle de définition de la fonction de forme résultante en prenant comme premier point valide le point dont la hauteur est le maximum des plus petites hauteurs admissibles des conteneurs considérés. L'algorithme est alors le suivant :

```
Pour chaque fils faire
  Si le fils est un module
     compose(fils)
     rotation(fils)

  liste_AVL <- ajouter l'AVL du fils
Fin pour

min <- plus grande des plus petites hauteurs parmi tous les bipoints de tous les AVL de liste_AVL
       c'est la borne minimale que doivent partager toutes les fonctions de forme
```

```
liste_bipoints <- liste ordonnée (suivant la hauteur croissante) de tous les bipoints de l'AVL de tous les fils

Pour tous les bipoints BP de liste_bipoints à partir du bipoint dont la hauteur est >= à  min
   hauteur_BP = hauteur du bipoint BP examiné

   Pour tout AVL de liste_AVL
      somme_largeur = somme_largeur + largeur obtenue lorsque la contrainte hauteur_BP est appliquée à AVL
                      somme_largeur + AVL.DXeff(hauteur_BP)
   Fin pour

   ajout du bipoint (hauteur_BP,somme_largeur)

Fin pour
```

L'algorithme de composition pour un conteneur vertical est symétrique, il suffit de remplacer la somme des largeurs par la somme des hauteurs et la hauteur par la largeur.

6.6.3 Exemple

On considère la composition de deux fonctions de forme dans le cas d'un conteneur horizontal.

Le graphe du haut de la figure 6.6 représente deux fonctions de forme de deux dispositifs élémentaires (rouge et bleu), le graphe du bas représente la fonction de forme résultant de la composition. Le domaine de validité des deux fonctions est différent, les surfaces hachurées sont des zones inaccessibles.

La composition a lieu en plusieurs étapes :

- Les points de chaque fonction de forme sont fusionnés en une seule liste de bipoints triés par hauteur croissante. Cette liste est examinée dans l'ordre croissant.
- La première valeur de la liste est $DY^1_{eff_1}$ du dispositif élémentaire 1. Elle ne fait pas partie du domaine de validité du dispositif élémentaire 2 et est donc éliminée.
- La valeur suivante de la liste est $DY^2_{eff_1}$. Elle fait partie des domaine de validité des deux dispositifs élémentaires. Cette contrainte de hauteur $DY^2_{eff_1}$ appliquée au dispositif élémentaire 1 donne la hauteur effective $DY^1_{eff_1}$ et appliquée au dispositif élémentaire 2 la hauteur effective $DY^2_{eff_1}$. La hauteur $DY^2_{eff_1}$ est conservée. Pour la hauteur retenue, les largeurs correspondantes des deux dispositifs élémentaires sont DX^1_1 et DX^2_1. Le nouveau point de la fonction de forme résultante à ajouter est $(DX^1_1 + DX^2_1, DY^2_{eff_1})$, ajout suivant la hauteur).
- La valeur suivante de la liste est $DY^1_{eff_2}$. Pour cette contrainte de hauteur, les hauteurs effectives du dispositif élémentaire 1 et du dispositif élémentaire 2 sont respectivement $DY^1_{eff_2}$ et $DY^2_{eff_1}$. La plus grande hauteur $DY^1_{eff_2}$ est retenue. Pour la hauteur retenue, les largeurs correspondantes sont DX^1_2 et DX^2_1. Le nouveau point de la fonction de forme a pour coordonnées $(DX^1_2 + DX^2_1, DY^1_{eff_2})$.
- Le processus est répété jusqu'à épuisement de la liste de bipoints.

On remarquera une augmentation du nombre de points de la fonction de forme résultant de la composition par rapport aux nombres de points initiaux des fonctions des deux dispositifs élémentaires. Le domaine de validité est l'intersection des deux domaines de validité initiaux.

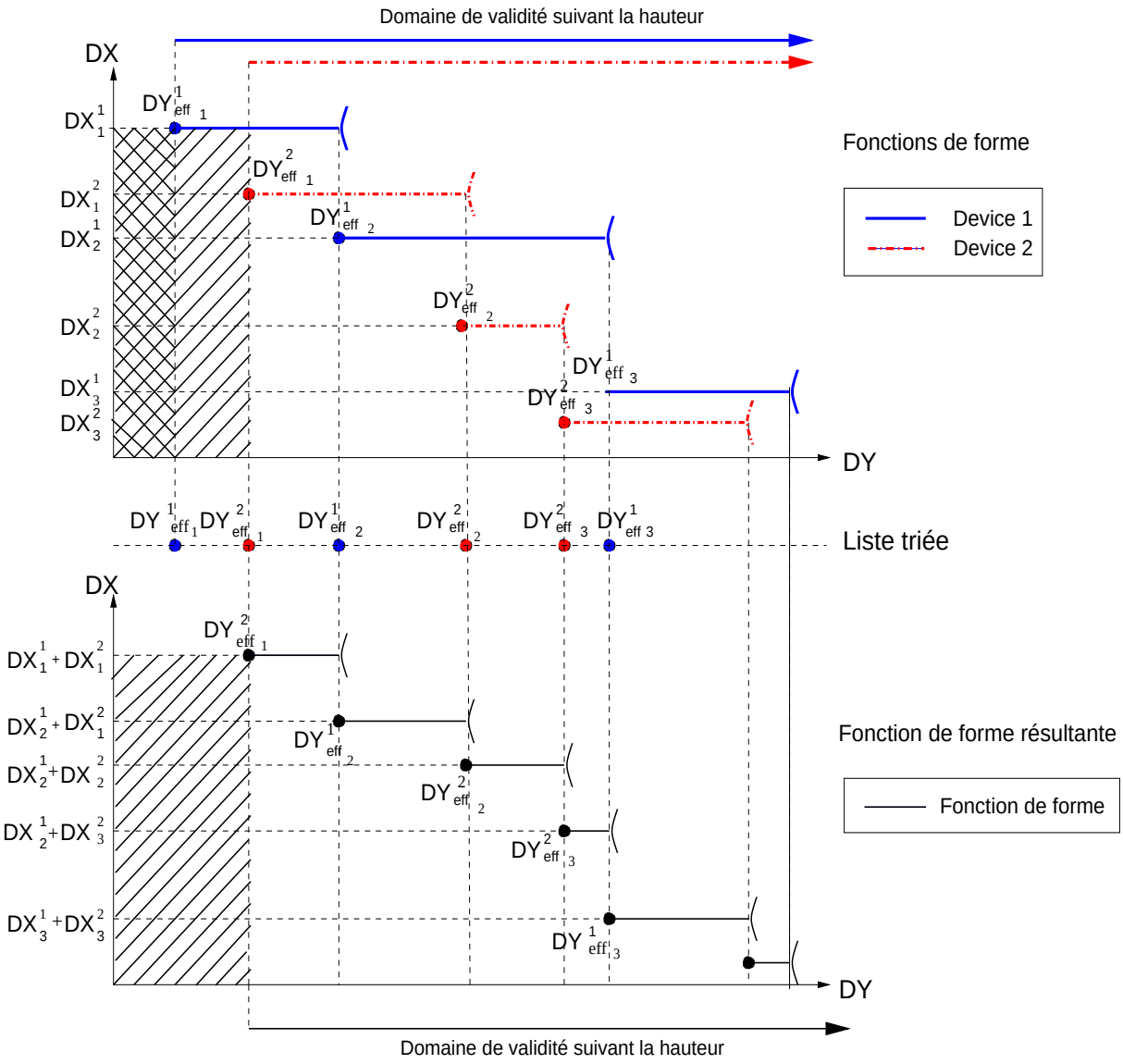


FIG. 6.6: Composition de deux fonctions de forme

6.7 Conclusion

Nous avons démontré dans ce chapitre qu'il était possible de construire, de façon totalement automatique, un placement optimisé des différents dispositifs élémentaires constituant un module. Pour une contrainte de hauteur (ou de largeur) maximale, ce placement est optimal du point de vue de la surface totale occupée quel que soit le dimensionnement électrique, et donc la surface occupée par chacun des dispositifs élémentaires.

Ce résultat s'appuie sur deux hypothèses :

- On dispose, pour chacun des dispositifs élémentaires, d'une fonction de forme monotone décroissante.

- On dispose, pour chacun des modules instanciés dans l'arbre des modules, de directives de placement relatif respectant la technique du « slicing tree »

Cette technique de génération automatique d'un placement optimisé est une aide importante pour simplifier le travail de conception d'un générateur de module.

Chapitre 7

Prise en compte des capacités parasites dans la phase de dimensionnement électrique

Sommaire

7.1 Capacités parasites 64
7.1.1 Routage 64
7.1.2 Dispositifs élémentaires 65
7.2 Interaction des vues physiques et électriques 66
7.2.1 Calcul des capacités parasites 66
7.2.2 Prise en compte des capacités parasites 68
7.3 Conclusion 69

Il existe de nombreux éléments parasites liés à la réalisation physique des masques, tant au niveau des dispositifs élémentaires (capacités parasites internes aux dispositifs élémentaires) qu'au niveau du routage (capacités parasites de routage), éléments qui influent sur le dimensionnement électrique. La méthodologie utilisée par CAIRO+ permet d'en tenir compte.

Une interaction fine entre le dimensionnement électrique et la réalisation physique ne peut avoir lieu sans extraction du dessin des masques à moins d'employer des estimations. CAIRO+ utilise un procédé mixte : dans un premier temps, le dimensionnement électrique permet d'évaluer certaines grandeurs physiques au niveau des dispositifs élémentaires (capacités de source, drain...), puis la génération automatique du dessin des masques permet le calcul précis des capacités parasites de routage sans extraction.

7.1 Capacités parasites

7.1.1 Routage

Le routage physique consiste à relier entre eux (et aux connecteurs externes du module père) les différents connecteurs physiques des modules appartenant à un même niveau hiérarchique.

Le routage est dit symbolique, car les dimensions des éléments manipulés (hauteur et largeur des modules instanciés) ne sont pas connues. Les procédures de routage, tout comme les procédures servant à ajouter des connecteurs physiques, sont conçues de manière à permettre un placement et un routage relatif. Il s'agit d'un routage canal, l'espace inter-modules servant à définir les canaux.

La première action du concepteur consiste à ajouter les connecteurs externes du module père sur les bords de la boite d'aboutement.

Le principe du routage procédural consiste à tracer des segments dont les extrémités s'appuient sur des connecteurs ou sur des points d'articulation (ou points d'ancrage) que le concepteur utilise pour rendre le routage relatif. La figure 7.1 donne un exemple de routage procédural où une interconnexion est composée de deux fils. Le premier est un fil à deux segments, le second un fil à quatre segments.

La figure 7.2 présente un détail du fil à quatre segments.

Chaque fil de routage, chaque via, possède une capacité par rapport au substrat dont il faut tenir compte lors du dimensionnement électrique.

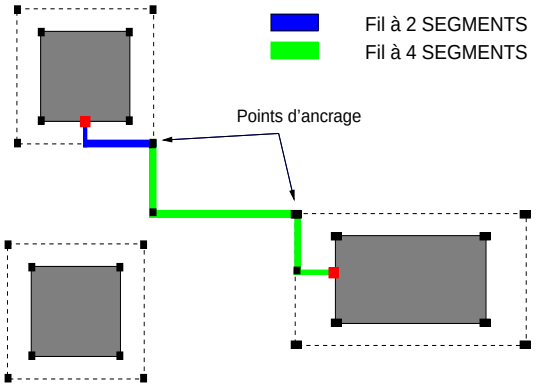


FIG. 7.1: Exemple de routage procédural

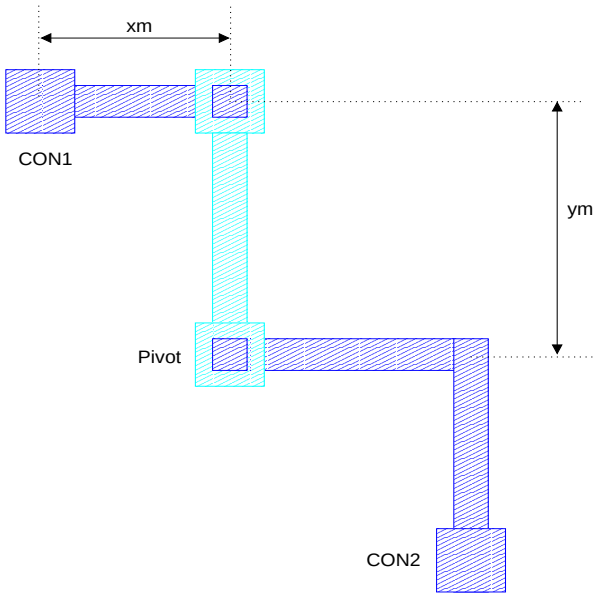


FIG. 7.2: Fil à quatre segments

7.1.2 Dispositifs élémentaires

Au sein même des dispositifs élémentaires, il existe des capacités parasites, non seulement au niveau du routage interne mais aussi au niveau des diffusions (figure 7.3 pour un transistor NMOS) et de toutes les couches.

La plupart du temps, le modèle électrique utilisé pour modéliser le comportement des dispositifs élémentaires prend en compte les capacités parasites. Les fonctions de la calculette de CAIRO+ permettent de calculer de telles capacités parasites.

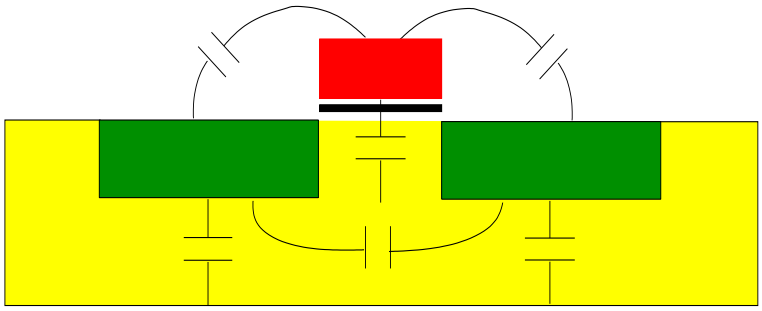


FIG. 7.3: Quelques capacités parasites pour un transistor NMOS

7.2 Interaction des vues physiques et électriques

Jusqu'à présent, le flot de conception décrit s'est attaché au dimensionnement électrique, en vue de produire des informations physiques à destination du dessin des masques. Il est cependant nécessaire de prendre en compte les informations parasites issue du dessin des masques, informations connues seulement après la fin du dimensionnement électrique lors de la génération physique. L'interaction des vues physiques et électriques ne peut prendre place qu'au sein d'une boucle.

7.2.1 Calcul des capacités parasites

Capacités parasites de routage par rapport au substrat

Afin d'éviter la phase d'extraction après la génération du dessin des masques, chaque fonction de tracé de segment utilisée lors du routage procédural rétro-annote le schéma associé en cours de dimensionnement (figure 7.4). Les dimensions physiques et les données technologiques étant connues à la fin de la procédure de placement, il est possible de calculer assez précisément les valeurs des capacités parasites dans la phase de routage.

Le concepteur du générateur dispose de fonctions d'accès rendant les valeurs des capacités parasites associées à chaque nœud électrique.

Capacités parasites pour un dispositif élémentaire, dimensionnement électrique

Le dimensionnement électrique des dispositifs élémentaires repose sur une bibliothèque de fonctions incluant les modèles électriques du dispositif représenté (modèle BSIM3V3 pour un transistor par exemple). Si les prototypes des fonctions proposées sont indépendantes du procédé cible (il s'agit d'une interface commune d'interrogation), les résultats renvoyés par ces fonctions sont évidemment fortement dépendants du procédé de fabrication choisi et des paramètres technologiques fournis par le fondeur.

Ces fonctions de la bibliothèque calculette sont utilisées par les dispositifs élémentaires lors du dimensionnement électrique. D'autres fonctions de cette même bibliothèque

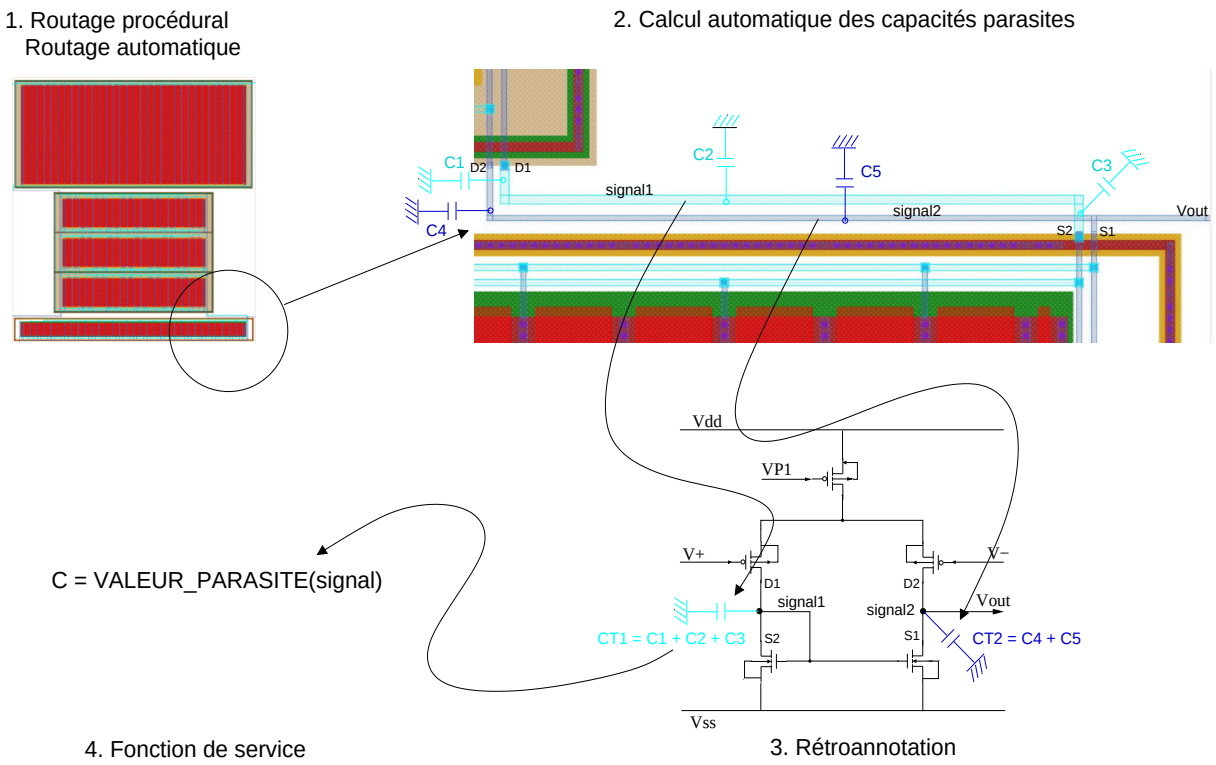


FIG. 7.4: Rétro-annotation des capacités parasites de routage et utilisation

permettent de calculer les capacités parasites associées aux drains, grilles et sources d'un transistor en fonction de ses caractéristiques géométriques[Tsi99][CH99] :

```
CAIRO_MOS_CGS(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CGB(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CGD(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CBS(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CBD(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CSD(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CM (Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CMX(Tr_name,Temp,W,L,VGS,VDS,VBS)
CAIRO_MOS_CMB(Tr_name,Temp,W,L,VGS,VDS,VBS)
```

Tr_name est le nom du transistor
Temp est la température
W et **L** sont les largeurs et longueurs du transistor
VGS est la tension entre la grille et la source
VDS est la tension entre le drain et la source
VBS est la tension entre le substrat et la source

Les capacités CGS, CGB, CGD, CBS, CBD, CSD, CM, CMX et CMB font partie des paramètres de sortie des procédures de dimensionnement des dispositifs élémentaires

utilisant des transistors. Ces capacités parasites peuvent être utilisées pour calculer la capacité effective de tout nœud du circuit (figure 7.5).

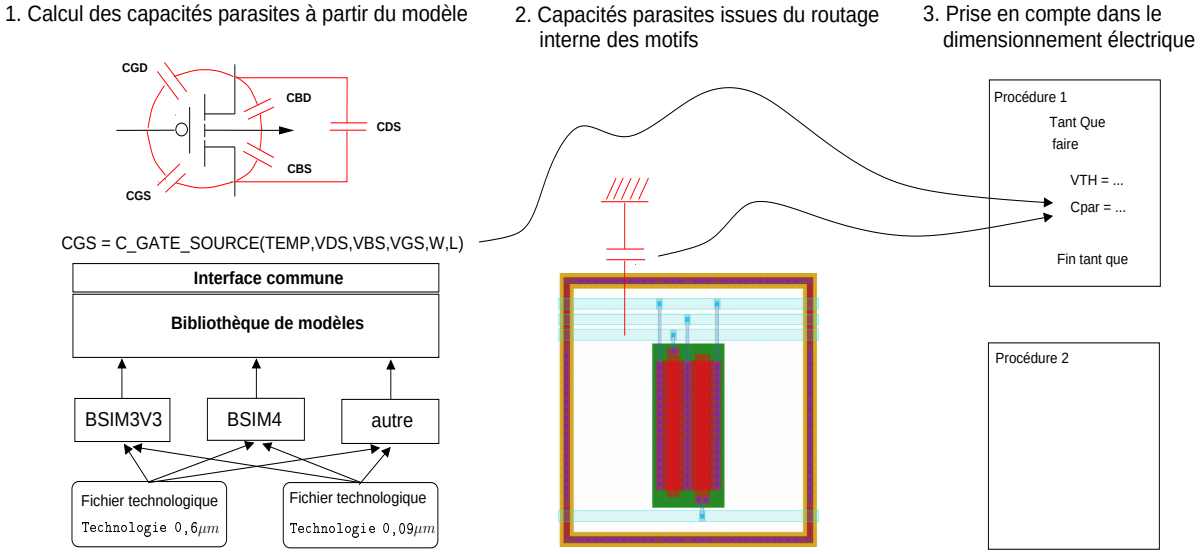


FIG. 7.5: Calcul des capacités parasites pour un dispositif élémentaire

7.2.2 Prise en compte des capacités parasites

Pour traiter le problème de l'interdépendance entre la phase de dimensionnement électrique (DSES) et les phases de génération du dessin des masques (PLACE et ROUTE), la solution consiste à exécuter plusieurs fois la séquence $DSES \rightarrow PLACE \rightarrow ROUTE$. Lors de la première itération, les calculs d'un dimensionnement électrique se font sans capacités parasites de routage (on ne dispose pas du dessin des masques, toutes les capacités parasites sont considérées comme étant nulles) et avec un nombre de doigts M égal à 1 (pour des transistors). Les capacités parasites liées aux transistors ne sont donc pas les valeurs exactes issues du dessin des masques. Il s'agit de valeurs initiales permettant d'effectuer les calculs sans avoir à sa disposition le dessin des masques, qui ne sera produit qu'après la phase PLACE. Les itérations suivantes permettent de recommencer le dimensionnement, mais en tenant compte des capacités parasites (figure 7.6).

Cette prise en compte se fait avec un décalage d'une itération : le DSES courant travaille avec les valeurs produites par le PLACE précédent. La différence peut être grande, surtout au niveau du nombre de repliements. Après un petit nombre d'itérations, le système finit par se stabiliser grâce à la stricte décroissance de la fonction de forme :

La stabilité du système est obtenue lorsque le nombre de repliements de chaque dispositif élémentaire ne varie plus entre deux itérations successives.

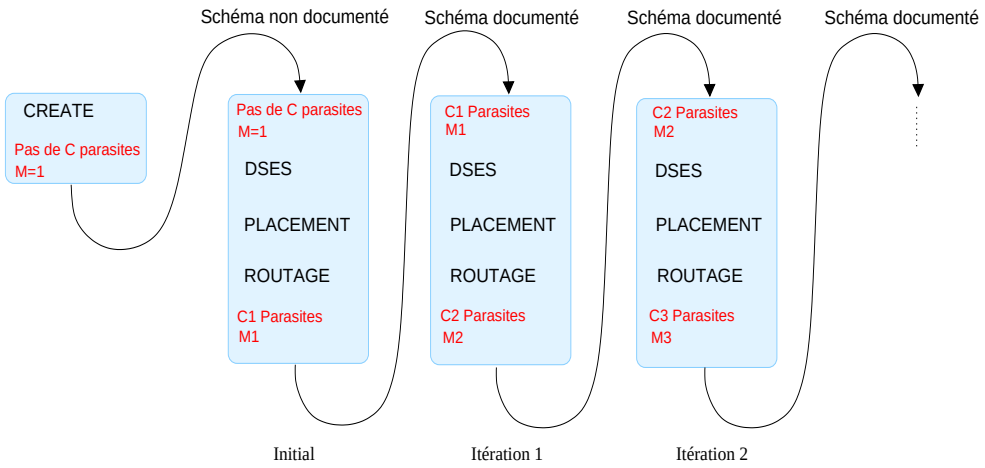


FIG. 7.6: Boucle pour la prise en compte des capacités parasites

La fonction de forme attachée aux dispositifs élémentaires étant, par construction, strictement monotone décroissante, il suffit d'examiner la variation de la hauteur et de la largeur du module racine :

La stabilité du système est obtenue lorsque la hauteur et la largeur du module racine ne varient plus entre deux itérations successives.

Les valeurs des capacités parasites sont stabilisées, les aires ne variant plus.

Le tableau 7.1 montre l'évolution des différents paramètres lors d'une boucle de relaxation appliquée à un OTA de type P composé d'un transistor BIAS, d'une paire différentielle DP et d'un miroir de courant CM. Les spécifications qui concernent le produit gain-bande GBW, la marge de phase PM et le facteur de forme sont les suivantes :

1. $GBW \geq 65Mhz$
2. $76 \leq PM \leq 96$
3. Le facteur de forme imposé est le carré

On note M le nombre de repliements des transistors, W et L leurs largeurs et longueurs. On note DY et DX la hauteur et la largeur du circuit obtenu.

La stabilisation est obtenue après trois itérations.

7.3 Conclusion

La co-conception physique-électrique dans CAIRO+ fait appel d'une part à une bibliothèque de modèles électriques capables de caractériser les capacités parasites lors d'un dimensionnement électrique sans recourir à une extraction des masques, d'autre part à un ensemble de fonctions de routage pour calculer de façon dynamique les capacités de routage et de rétro-annoter le schéma.

TAB. 7.1: Stabilisation des paramètres en technologie 0,6 μm

Dispositif	Paramètres	Initial	Itération 1	Itération 2	Itération 3
BIAS	M utilisé	1	32	32	32
	W (μm)	1684,17	1532,6	1451,32	1451,32
	L (μm)	2,52	2,52	2,455	2,455
DP	M utilisé	1	16	32	32
	W (μm)	409,23	411,17	403,10	403,10
	L (μm)	2,52	2,52	2,455	2,455
CM	M utilisé	1	16	16	16
	W (μm)	86,12	89,89	88,05	88,05
	L (μm)	2,52	2,52	2,455	2,455
	GBW (Mhz)	69,19	69,11	69,16	69,16
	PM (degré)	76,01	75,56	76,00	76,00
	DY (μm)	106,98	127,26	123,9	123,9
	DX (μm)	118,66	118,44	116,36	116,36
	Paramètres	Initial	Itération 1	Itération 2	Itération 3

La convergence du dimensionnement électrique tenant compte des capacités parasites est automatiquement assurée par une boucle de relaxation, ce qui permet de se passer des longues phases de simulations après extraction.

Chapitre 8

CAIRO+ : une plateforme de développement

Sommaire

8.1 Concepts généraux 72
8.1.1 Programme exécutable 73
8.1.2 Phases du flot de conception et fonctions associées 73
8.2 Création des structures de données 74
8.2.1 Déclaration des composants 74
8.2.2 Patron électrique 75
8.2.3 Patron physique 76
8.2.4 Interface fonctionnelle 79
8.3 Exploration du domaine de conception : DSES 81
8.3.1 Corps d'une procédure 81
8.3.2 Communication descendante : passage des arguments 81
8.3.3 Communication ascendante : transmission des résultats 83
8.3.4 Gestion des échecs 86
8.4 Formation d'un exécutable 90
8.5 Résumé 94

Les concepts exposés dans les chapitres précédents trouvent ici leur développements détaillés. Le nouveau flot de conception proposé est une méthodologie que l'analogicien doit suivre lors de l'écriture des générateurs. Elle est sous-tendue par un langage qui le guide tout en lui donnant l'occasion de s'exprimer librement. CAIRO+ est donc plus qu'un idiome analogique du langage C++ dont il est issu, c'est une plateforme de développement où chaque étape est matérialisée par l'écriture d'une fonction.

8.1 Concepts généraux

Le flot de conception de CAIRO+ est composé de quatre phases, rappelées en figure 8.1. Chaque phase est associée soit à une fonction dont l'écriture est à la charge du concepteur de générateur, soit liée à l'appel d'une fonction particulière du langage CAIRO+.

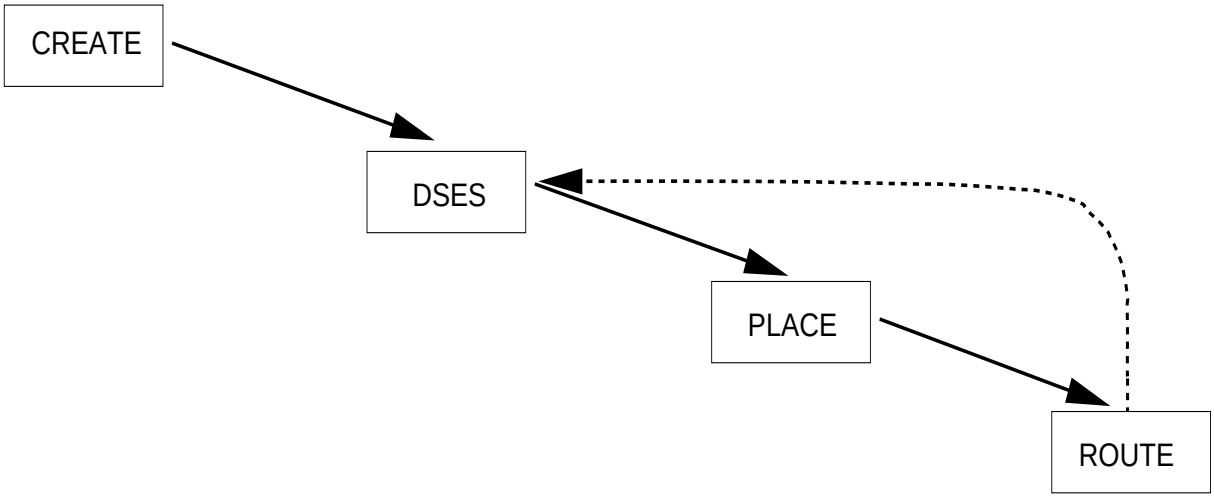


FIG. 8.1: Étapes de conception et fonctions associées

Pour chaque générateur de module, le concepteur doit écrire trois fonctions :

- La fonction de création des instances de la phase CREATE, nommée CAIRO_CREATE ;
- La fonction d'exploration du domaine de conception rassemblant les procédures de dimensionnement électrique de la phase DSES, nommée CAIRO_DSES ;
- La fonction de routage des instances, nommée CAIRO_LAYOUT (phase ROUTE).

8.1.1 Programme exécutable

Les fonctions associées à un dispositif élémentaire ou à un module sont écrites indépendamment les unes des autres. Lors de l'écriture d'un générateur, le concepteur n'a donc pas à se soucier de la gestion des appels, il doit simplement écrire les trois fonctions définies ci-dessus. L'amorce est l'exécutable principal, qui reste obligatoire. Il a la charge d'initier la création du circuit et de mener à bien l'exploration du domaine de conception.

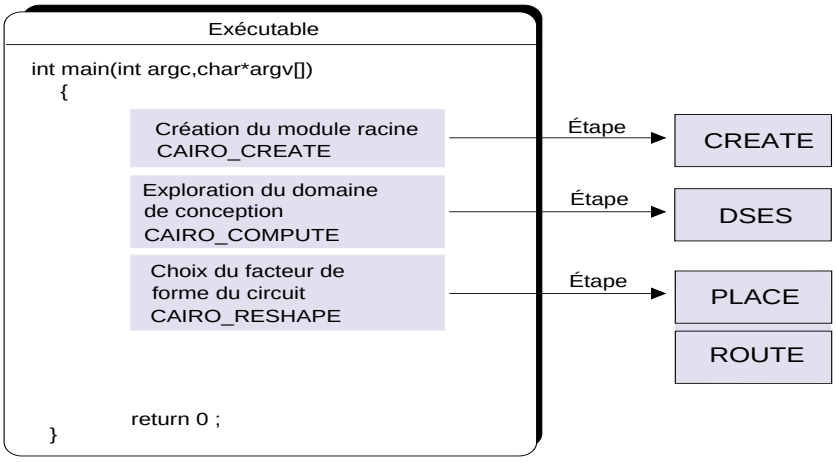


FIG. 8.2: Un exécutable

L'exécutable se compose de trois blocs principaux : la création du module racine, l'exploration du domaine de conception et le choix du facteur de forme (figure 8.2).

8.1.2 Phases du flot de conception et fonctions associées

On présente ici le prototype des fonctions associées aux phases décrites. Elles constituent l'interface (au sens système du terme) de tout générateur.

- La phase de création construit les structures de données. La fonction remplissant cet office est nommée **CAIRO_CREATE**. La déclaration est de la forme :

```
CAIRO_BEGIN_CREATE(generator_name,optional parameters)
  // code
CAIRO_END_CREATE(generator_name)
```

- La phase d'exploration du domaine de conception permet d'interroger le générateur sur la faisabilité de certaines spécifications. Elle est nommée **CAIRO_DSES** et est de la forme

```
CAIRO_BEGIN_DSES(generator_name)
  // code
CAIRO_END_DSES(generator_name)
```

- La phase dessin des masques est circonscrite au routage des modules. Sa déclaration est :

```
CAIRO_BEGIN_LAYOUT(generator_name,optional parameters)
  // code
CAIRO_END_LAYOUT(generator_name)
```

8.2 Création des structures de données

La fonction de création est destinée à définir l'interface électrique, le schéma électrique non dimensionné (*netlist template*), le placement relatif des éléments (*layout template*), la déclaration des paramètres électriques du module, et la déclaration des procédures de calcul (*Procedural Service Interface*) et leurs paramètres associés.

Le prototype de la fonction de création est redonné ici :

```
CAIRO_BEGIN_CREATE(generator_name,optional parameters)
  // code
CAIRO_END_CREATE(generator_name)
```

generator_name est un nom caractéristique du module générique

Les **paramètres optionnels** sont au libre choix du concepteur. Ils lui permettent de spécifier lors de l'instanciation la valeur de paramètres structurels (équivalent des *generic* du VHDL). Le lecteur pourra consulter le mode d'emploi pour plus de détails.

8.2.1 Déclaration des composants

La situation est différente suivant qu'il s'agit d'un module (il fait appel à d'autres modules et/ou à des dispositifs élémentaires) ou d'un dispositif élémentaire.

L'instanciation est une déclaration d'intention d'utiliser les éléments. Ils sont instanciés sans pour autant posséder les relations (connexions, placement) qui seront définies dans les patrons électrique et physique. Le module instancie d'autres modules ou des dispositifs élémentaires. L'instanciation est réalisée par la fonction :

```
CAIRO_CREATE(char *library_name,char *generator_name,
             char *instance_name[,optional parameters]) ;
```

library_name est le nom de la bibliothèque contenant le générateur

generator_name est le nom du générateur

instance_name est le nom d'instance du dispositif élémentaire ou du module instancié

optional parameters sont les paramètres dépendant de la déclaration du générateur (pour cela il faut consulter la documentation associée à chaque bibliothèque). Chaque paramètre est séparé du précédent par une virgule.

L'exemple suivant, tiré d'un OTA de la bibliothèque libOTA de CAIRO+, montre l'instanciation d'un dispositif élémentaire transistor simple, d'un dispositif élémentaire paire différentielle et d'un dispositif élémentaire miroir de courant :

```
CAIRO_BEGIN_CREATE(OTA,char type)

  // Transistor simple BIAS de type P
  CAIRO_CREATE("libMOS","TR_MOS","BIAS","P5",TRANSP) ;

  // Paire différentielle DP de type P
  CAIRO_CREATE("libMOS","DP_MOS","DP","P1","P2",TRANSP) ;

  // Miroir de courant simple CM de type N
  CAIRO_CREATE("libMOS","CM_MOS","CM","N3","N4",TRANSN) ;

CAIRO_END_CREATE(OTA)
```

8.2.2 Patron électrique

Le patron électrique (*netlist template*) définit les relations électriques entre les instances. Les connecteurs électriques externes (ceux du module généré) et internes (ceux des instances) sont reliés par les signaux électriques.

Interface électrique

L'interface électrique définit la partie visible d'un module, ses connecteurs électriques. La déclaration en CAIRO+ est réalisée grâce à la fonction

```
CAIRO_IO(char *connector_name,face[,face serie]) ;
```

connector_name est le nom du connecteur électrique
face est la face physique de sortie du (des) connecteur(s) physique(s) associé(s) au connecteur électrique. face peut valoir CP_NORTH, CP_SOUTH, CP_EAST, CP_WEST. Cette notion est à rapprocher du patron physique qui est présenté par la suite. Un connecteur électrique peut être associé à plusieurs connecteurs physiques. Ceux-ci possèdent une direction de sortie sur une face du module généré (qui physiquement est un rectangle).

Signaux

Les signaux définissent les connexions entre les instances. La fonction CAIRO+ utilisée pour connecter un composant est :

```
CAIRO_CONNECT(char *instance_name,"connector => signal"[,connexion serie])
```

instance_name est le nom d'instance défini lors de l'appel à CAIRO_CREATE (troisième argument)
connector => **signal** est le couple connecteur-signal

connexion serie est la liste des couples connecteurs-signaux de l'instance considérée. Le nombre de signaux à relier dépend du module instancié. L'ordre d'apparition des couples connecteur-signal est quelconque.

On considère le même OTA que précédemment, OTA dont les dispositifs élémentaires sont connectés tels que le montre la figure 8.3 (les faces de sortie des connecteurs physiques ne sont pas représentées). Le code correspondant en CAIRO+ est :

```
CAIRO_BEGIN_CREATE(OTA,char type)

  // --------- Instanciations
  // Transistor simple BIAS de type P
  CAIRO_CREATE("libMOS","TR_MOS","BIAS","P5",TRANSP) ;

  // Paire différentielle DP de type P
  CAIRO_CREATE("libMOS","DP_MOS","DP","P1","P2",TRANSP) ;

  // Miroir de courant simple CM de type N
  CAIRO_CREATE("libMOS","CM_MOS","CM","N3","N4",TRANSN) ;

  // --------- Connecteurs électriques
  CAIRO_IO("VP1" ,CP_WEST ) ; // Tension de polarisation
  CAIRO_IO("VIN1",CP_WEST ) ; // Entrée différentielle +
  CAIRO_IO("VIN2",CP_WEST ) ; // Entrée différentielle -
  CAIRO_IO("VOUT",CP_EAST ) ; // Sortie
  CAIRO_IO("VDD" ,CP_EAST ) ; // Alimentation
  CAIRO_IO("VSS" ,CP_SOUTH) ; // Alimentation

  // --------- Connexions: la netlist
  // BIAS : drain gate source (bulk relie a la source en interne)
  CAIRO_CONNECT("BIAS","drain => VA","gate => VP1","source => VDD");

  // DP : drain1 drain2 gate1 gate2 source (bulk relie a la source en interne)
  CAIRO_CONNECT("DP","drain1 => VOUT","drain2 => VB","gate1 => VIN2","gate2 => VIN1","source => VA") ;

  // CM : drain1 drain2 source (bulk relie a la source en interne)
  CAIRO_CONNECT("CM","drain1 => VB","drain2 => VOUT","source => VSS");

CAIRO_END_CREATE(OTA)
```

8.2.3 Patron physique

Constitution d'un conteneur

Les conteneurs sont formés de deux boites rectangulaires (figure 8.4) :

- Une boite de garde délimitant la partie interne du module ou du dispositif élémentaire.
- Une boite englobante délimitant un espace (noté CP_NORTH, CP_SOUTH, CP_EAST et CP_WEST) utilisé pour le routage canal.

Les conteneurs n'ont pas d'existence physique, il servent à partitionner l'espace physique où les modules et les dispositifs élémentaires sont placés et routés.

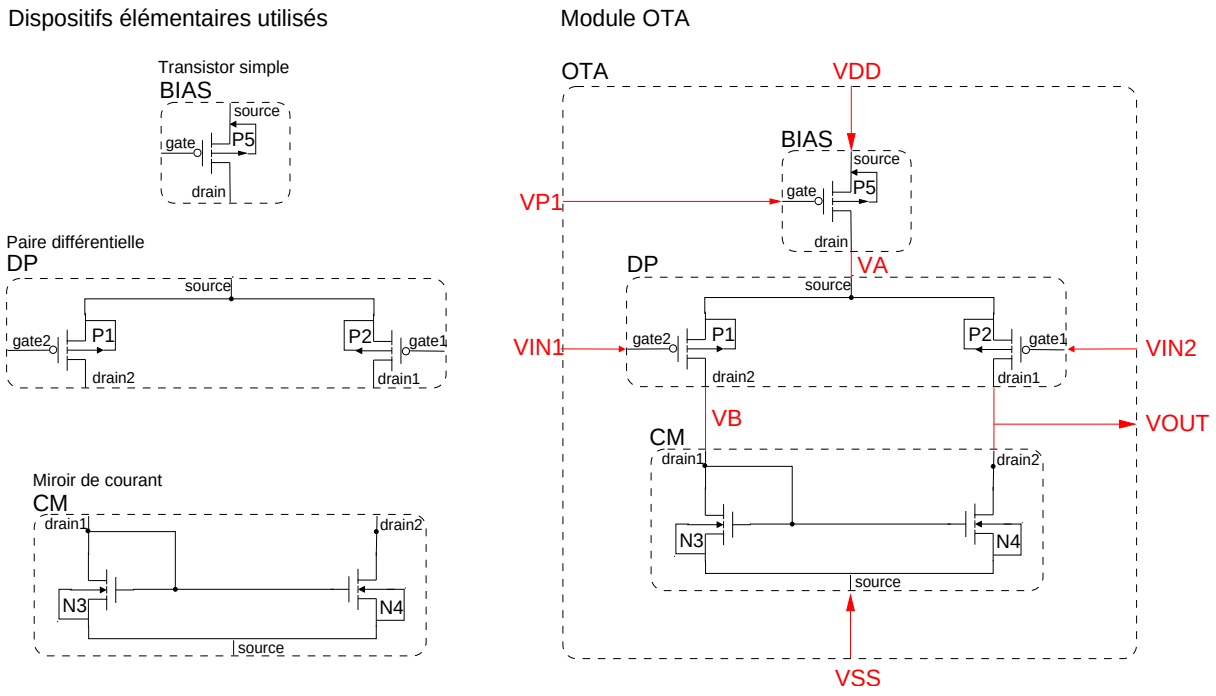


FIG. 8.3: Patron électrique de l'OTA de type P

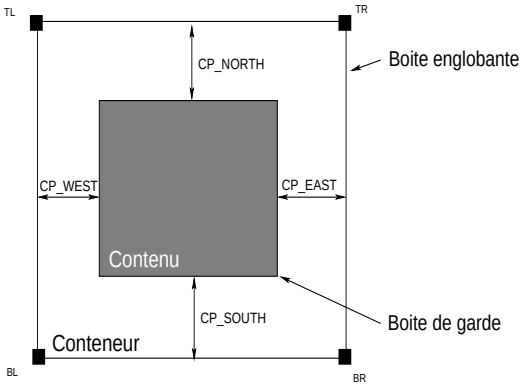


FIG. 8.4: L'environnement du conteneur

Déclaration des conteneurs dans CAIRO+

La déclaration d'un conteneur horizontal s'effectue en utilisant la fonction :

```
CAIRO_HORIZONTAL_CONTAINER(char *name[,char *name,symetry,north,
                                    south,east,west[,container serie]])
```

name est le nom du conteneur. name est suivi d'une suite de ["name",symetry,north,south,east,west] pour chacun des objets instanciés (module, dispositif élémentaire ou autre conteneur).

symetry est la symétrie d'instanciation. Elle vaut NOSYM pour aucune symétrie, ROT_P pour une rotation anti-horaire de 90 degrés, ROT_M pour une rotation horaire de 90 degrés, SYM_X pour une symétrie d'axe X, SYM_Y pour une symétrie d'axe Y, SYMXY pour une symétrie d'axe X suivie d'une symétrie d'axe Y, SY_RP pour une symétrie d'axe X suivie d'une rotation anti-horaire de 90 degrés, SY_RM pour une symétrie d'axe X suivie d'une rotation horaire de 90 degrés.

north, **south**, **east**, **west** sont les environnements nord, sud, est et ouest du conteneur inséré dans le conteneur horizontal.

Suivant le même modèle, un conteneur vertical a pour définition :

```
CAIRO_VERTICAL_CONTAINER(char *name[,char *name,symetry,north,
                                    south,east,west[,container serie]])
```

Les éléments placés dans un conteneur horizontal sont alignés de la gauche vers la droite. Ceux placés dans un conteneur vertical sont empilés du bas vers le haut. Le module dont on est en train d'écrire la fonction de création doit à la fin posséder son propre conteneur.

L'exemple suivant montre l'utilisation du conteneur vertical pour un OTA simple (figure 8.5) :

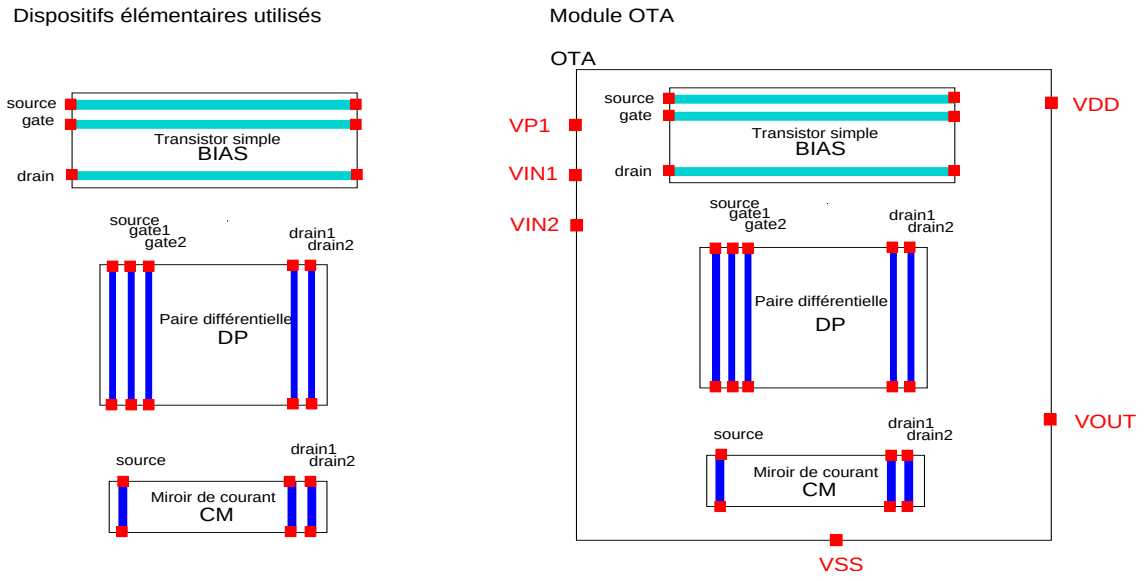


FIG. 8.5: Patron physique de l'OTA de type P

Le code CAIRO+ correspondant est :

```
CAIRO_BEGIN_CREATE(OTA,char type)

  // --------- Instanciations
  // Transistor simple BIAS de type P
  CAIRO_CREATE("libMOS","TR_MOS","BIAS","P5",TRANSP) ;

  // Paire différentielle DP de type P
  CAIRO_CREATE("libMOS","DP_MOS","DP","P1","P2",TRANSP) ;

  ...

  // Miroir de courant simple CM de type N
  CAIRO_CREATE("libMOS","CM_MOS","CM","N3","N4",TRANSN) ;

  // --------- Patron physique
  CAIRO_VERTICAL_CONTAINER(name,"CM"   ,NOSYM,3 * PITCH,2 * PITCH,SD_NWELL_PTIE,SD_NWELL_PTIE,
                               "DP"   ,NOSYM,2 * PITCH,SD_NTIE  ,SD_NTIE      ,SD_NTIE      ,
                               "BIAS",NOSYM,2 * PITCH,SD_NTIE  ,PITCH * 2    ,PITCH * 2     ) ;

CAIRO_END_CREATE(OTA)
```

8.2.4 Interface fonctionnelle

Déclaration d'un paramètre

Les paramètres utilisés lors de l'exploration du domaine de conception doivent être déclarés dans la fonction de création. La fonction est de la forme :

```
CAIRO_DECLARE_PARAM(char *parameter_name)
```

parameter_name est le nom du paramètre

Type de paramètre

Pour chaque procédure, un paramètre peut être une entrée (il est donc fourni par le niveau supérieur), une sortie (la valeur est positionnée par le module courant) ou être en entrée-sortie (il est partiellement contraint). Le type d'un paramètre dépend de la procédure qui utilise ce paramètre (une procédure doit donc indiquer les paramètres utilisés, ainsi que leurs types). Les types sont notés respectivement :

```
CP_FIXED  (paramètre en entrée)
CP_NO     (paramètre en sortie)
CP_RANGE  (paramètre contraint par un intervalle)
CP_SUP    (paramètre contraint par une borne supérieure large)
CP_INF    (paramètre contraint par une borne inférieure large)
```

L'exemple suivant présente la déclaration des paramètres d'un OTA.

```
CAIRO_BEGIN_CREATE(OTA,char type)

  ...

  CAIRO_DECLARE_PARAM("TEMP"    ) ;
  CAIRO_DECLARE_PARAM("VEG_BIAS") ;
  CAIRO_DECLARE_PARAM("VEG_DP"  ) ;
  CAIRO_DECLARE_PARAM("L_INIT"  ) ;
  CAIRO_DECLARE_PARAM("VIN"     ) ;
  CAIRO_DECLARE_PARAM("VOUT"    ) ;
  CAIRO_DECLARE_PARAM("IBIAS"   ) ;
  CAIRO_DECLARE_PARAM("CLOAD"   ) ;

  CAIRO_DECLARE_PARAM("VP1"      ) ;
  CAIRO_DECLARE_PARAM("SR"       ) ;
  CAIRO_DECLARE_PARAM("GAIN"     ) ;

  CAIRO_DECLARE_PARAM("GBW"      ) ;
  CAIRO_DECLARE_PARAM("PM"       ) ;

  ...

CAIRO_END_CREATE(OTA)
```

Déclaration d'une procédure

La déclaration est la suivante :

```
CAIRO_DECLARE_PROCEDURE(char *procedure_name,class
                        [,char *parameter_name,type[,parameter serie]])
```

procedure_name est le nom de la procédure
class est la classe de la procédure, complète ou incomplète
parameter_name est le nom du paramètre attaché. Une procédure accepte un nombre quelconque de paramètres (quelque soit leurs types).
type définit le type de la contrainte.

L'exemple suivant montre la déclaration de deux procédures d'un OTA. La première, nommée VP1, est incomplète, la seconde (GBW) est complète car, à l'issue de son appel, l'OTA est dimensionné (tous ses transistors ont reçus des dimensions L et W).

```
CAIRO_BEGIN_CREATE(OTA,char type)

  ...

  // -------------------------
  // Déclaration des procédures
  // -------------------------

  CAIRO_DECLARE_PROCEDURE("VP1",CP_UNCOMPLETE,"TEMP",CP_FIXED,"VEG_BIAS",CP_FIXED,"VEG_DP",CP_FIXED,
                                              "L_INIT",CP_FIXED,"VIN",CP_FIXED,"VOUT",CP_FIXED,
                                              "IBIAS",CP_FIXED,"VP1",CP_NO) ;

  CAIRO_DECLARE_PROCEDURE("GBW",CP_COMPLETE,"TEMP",CP_FIXED,"CLOAD",CP_FIXED,
                                            "IBIAS",CP_FIXED,"SR",CP_NO,"GAIN",CP_NO,
                                            "GBW",CP_INF,"PM",CP_RANGE) ;
  ...

CAIRO_END_CREATE(OTA)
```

8.3 Exploration du domaine de conception : DSES

8.3.1 Corps d'une procédure

La déclaration de la procédure a été effectuée dans la fonction de création du générateur. L'implantation du corps de la procédure est réalisée dans la fonction DSES de la manière suivante :

```
CAIRO_BEGIN_PROCEDURE(char *procedure_name)
  // Calculs
END_PROCEDURE
```

procedure_name est le nom de la procédure

Il est possible d'implanter plusieurs procédures pour un même module, et l'ordre des procédures est non significatif :

```
CAIRO_BEGIN_DSES(generator_name)

  CAIRO_BEGIN_PROCEDURE("I(W)")
    // Code
  END_PROCEDURE

  CAIRO_BEGIN_PROCEDURE("I(L)")
    // Code
  END_PROCEDURE

CAIRO_END_DSES(generator_name)
```

8.3.2 Communication descendante : passage des arguments

Positionnement des paramètres

Le concepteur d'un générateur de niveau (N+1) utilisateur d'une procédure appartenant à un générateur de niveau N doit (si elle en possède) positionner ses paramètres avant de l'appeler (figure 8.6, point ①).

Les paramètres ayant été déclarés par la fonction CAIRO_DECLARE_PARAM, la valeur de la contrainte (valeur, borne ou intervalle) est imposée grâce aux fonctions suivantes :

- Pour une contrainte de type égalité :

```
CAIRO_SET_PARAM_VAL(char *instance_name,char *param_name,value)
```

instance_name est le nom de l'instance contenant le paramètre
param_name est le nom du paramètre auquel s'applique la contrainte
value est la valeur de la contrainte

- Pour une borne supérieure :

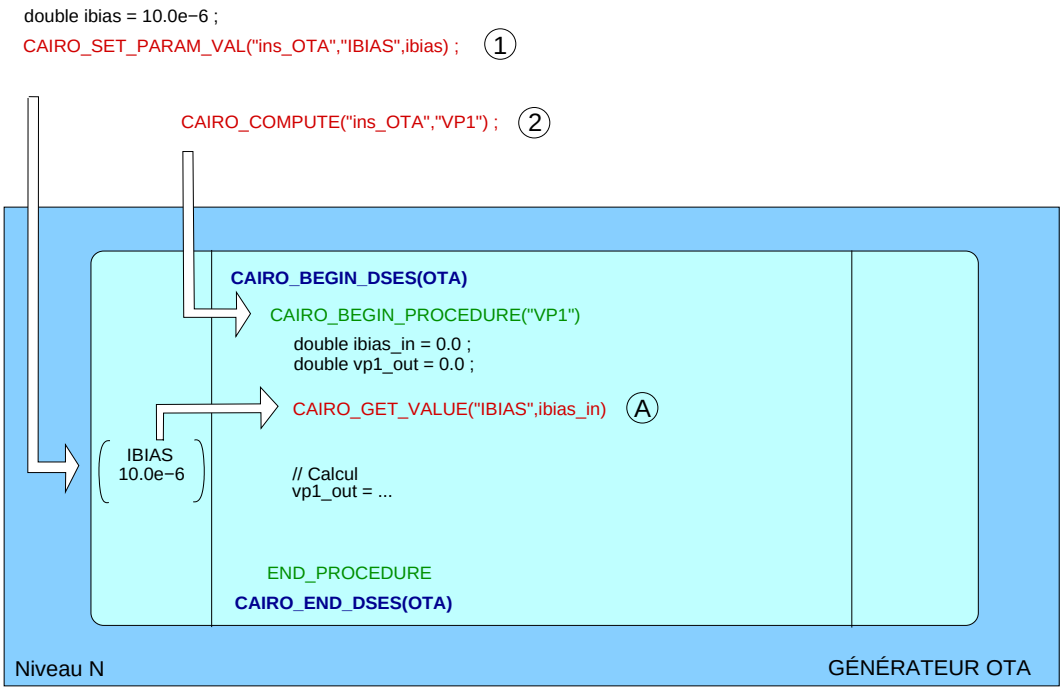


FIG. 8.6: Communication descendante pour un OTA

```
CAIRO_SET_PARAM_SUP(char *instance_name,char *param_name,value)
```

- Pour une borne inférieure :

```
CAIRO_SET_PARAM_INF(char *instance_name,char *param_name,value)
```

Une valeur est donnée aux bornes d'une contrainte de type intervalle en utilisant à la fois CAIRO_SET_PARAM_INF et CAIRO_SET_PARAM_SUP.

L'exemple suivant montre les contraintes appliquées aux paramètre d'un OTA.

```
double gbw       = 65.0e+06   ;
double pm_low    = 76.0       ;
double pm_high   = 90.0       ;

// Contrainte pour gbw
CAIRO_SET_PARAM_INF("OTA","GBW",gbw) ;

// Contrainte pour pm
CAIRO_SET_PARAM_INF("OTA","PM",pm_low) ;
CAIRO_SET_PARAM_SUP("OTA","PM",pm_high) ;
```

Appel de la procédure

Une procédure du générateur de niveau (N+1) peut appeler une procédure d'un générateur de niveau N en utilisant la fonction (figure 8.6, point ②) :

```
CAIRO_COMPUTE(char *instance_name,char *procedure_name) ;
```

instance_name est le nom d'instance du module à laquelle appartient la procédure
procedure_name est le nom de la procédure à appeler

Récupération des paramètres

Les deux précédentes actions sont menées par le générateur de niveau (N+1). La récupération des valeurs des paramètres contraints est effectuée par le générateur de niveau N lorsque la procédure est appelée. Le concepteur dispose des fonctions suivantes pour récupérer les valeurs des contraintes sur les paramètres d'entrée :

- Pour une contrainte de type égalité :

```
CAIRO_GET_VALUE(char *param_name,value)
```

param_name est le nom du paramètre auquel s'applique la contrainte
value est une variable C++ recevant la valeur du paramètre.

- Pour une borne supérieure :

```
CAIRO_GET_SUP(char *param_name,value)
```

- Pour une borne inférieure :

```
CAIRO_GET_INF(char *param_name,value)
```

Toutes les valeurs des paramètres dont a besoin la procédure de niveau N sont ainsi récupérées, ce qui lui permet de réaliser le calcul (service) associé (figure figure 5.5, point A). Elle peut éventuellement interroger le niveau hiérarchique (N-1) de la même manière.

8.3.3 Communication ascendante : transmission des résultats

Positionnement des paramètres de sortie

La fonction CAIRO_SET_VALUE réalise la première partie du passage de données dans le générateur de niveau N contenant la procédure (figure 8.7, point Ⓑ), la seconde étant réalisée par CAIRO_GET_PARAM au niveau (N+1) de l'appelant :

```
CAIRO_SET_VALUE(char *parameter_name,value)
```

parameter_name est le nom du paramètre
value est une variable C++ contenant la valeur du paramètre.

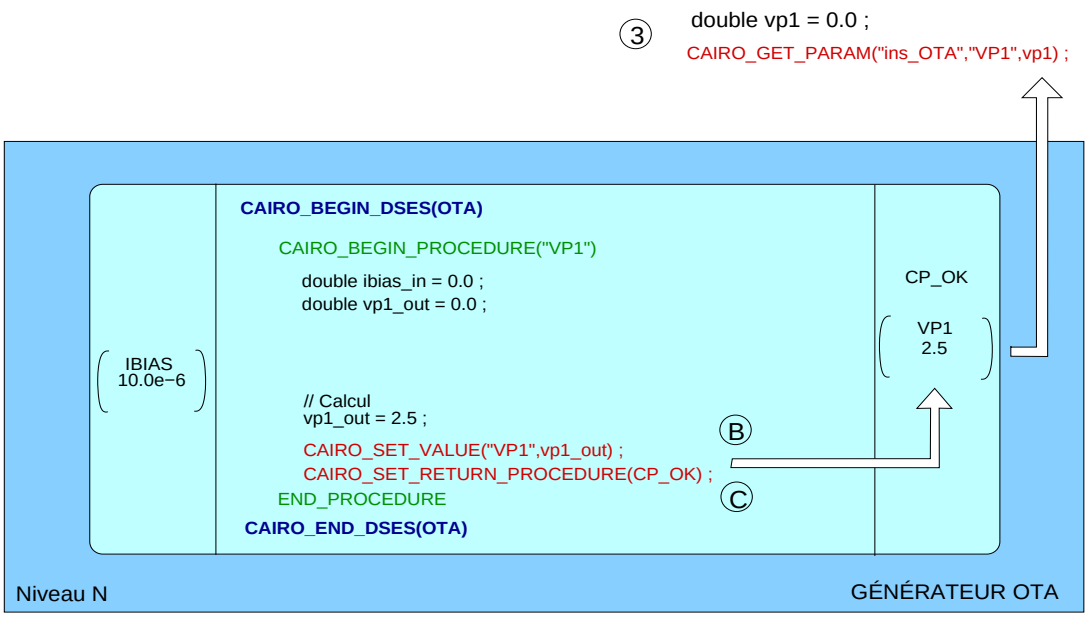


FIG. 8.7: Communication ascendante pour un OTA

Positionnement du code de retour de la procédure

Toute procédure possède un code de retour. Le concepteur utilise la fonction (figure 8.7, point Ⓒ) :

```
CAIRO_SET_RETURN_PROCEDURE(value)
```

value est un entier dont la valeur est libre (value ≥ 1). Le symbole prédéfini CP_OK est utilisé pour indiquer la réussite de la procédure.

Récupération des paramètres de sortie

Le concepteur utilise la fonction (figure 8.7, point ③) :

```
CAIRO_GET_PARAM(char *instance_name,char *parameter_name,value)
```

instance_name est le nom d'instance du module
parameter_name est le nom du paramètre demandé
value est une variable C++ recevant la valeur du paramètre.

L'exemple suivant montre un générateur de niveau (N+1) positionnant les paramètres de la procédure d'un OTA instancié (niveau N). La procédure est le calcul de la tension de polarisation (procédure « VP1 »). Celui-ci récupère ensuite la valeur associée à la tension demandée.

```
double temp      = 300.15     ;

double vin       = 1.65       ;
double vout      = 1.65       ;
double ibias     = 500.0e-6   ;
double ibias_cal = 0.0        ;
double vp1       = 0.0        ;
double cload     = 5.0e-12    ;
double veg_bias  = 0.2        ;
double veg_dp    = 0.2        ;

// ----- Positionnement des paramètres
CAIRO_SET_PARAM_VAL("OTA","TEMP"    ,temp    ) ;
CAIRO_SET_PARAM_VAL("OTA","IBIAS"   ,ibias   ) ;
CAIRO_SET_PARAM_VAL("OTA","VIN"     ,vin     ) ;
CAIRO_SET_PARAM_VAL("OTA","VOUT"    ,vout    ) ;
CAIRO_SET_PARAM_VAL("OTA","VEG_BIAS",veg_bias) ;
CAIRO_SET_PARAM_VAL("OTA","VEG_DP"  ,veg_dp  ) ;

// ----- Appel de la procédure
CAIRO_COMPUTE("OTA","VP1") ;

// ----- Récupération de vp1
CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;
```

Au niveau N, c'est à dire dans la procédure « VP1 » de l'OTA, l'implantation (raccourcie) montrant la récupération des paramètres, le calcul et le positionnement du paramètre VP1 en sortie est :

```
// --------------------- Procédure VP1
CAIRO_BEGIN_PROCEDURE("VP1")

  double temp       = 0.0 ; // Spécifications
  double vdd        = 0.0 ;
  double vss        = 0.0 ;
  double veg_bias   = 0.0 ;
  double veg_dp     = 0.0 ;
  double vin        = 0.0 ;
  double vout       = 0.0 ;
  double ibias_in   = 0.0 ;
  double vp1        = 0.0 ;

  // --- Récupération de la température
  CAIRO_GET_VALUE("TEMP",temp)

  // --- Récupération de IBIAS
  CAIRO_GET_VALUE("IBIAS",ibias_in)

  // --- Récupération de VIN
  CAIRO_GET_VALUE("VIN",vin)

  // --- Récupération de VOUT
  CAIRO_GET_VALUE("VOUT",vout)

  // --- Récupération de VEG_BIAS
  CAIRO_GET_VALUE("VEG_BIAS",veg_bias)

  // --- Récupération de VEG_DP
  CAIRO_GET_VALUE("VEG_DP",veg_dp)
```

```
  // ------------------------------------
  // --- Calculs
  // ------------------------------------

  ...

  // --- Positionnement de VP1
  CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;
  CAIRO_SET_VALUE("VP1",vp1) ;

END_PROCEDURE
```

Les paragraphes suivants sont consacrés au traitement des cas d'échecs lors de l'appel d'une procédure.

8.3.4 Gestion des échecs

Bloc de traitement du code de retour de la procédure

L'appel à la procédure de niveau N a entraîné le calcul d'un certain nombre de paramètres dont les valeurs sont renvoyées vers le niveau (N+1). Un code de retour attaché à la procédure exprime la réussite de l'appel ou la non validité des calculs ; hormis le code standard CP_OK utilisé pour la réussite, la valeur du code est définie par le concepteur du générateur de niveau N.

Le traitement du code de retour est réalisé par le concepteur du générateur de niveau (N+1) avec une structure classique *switch* du langage C :

```
switch(PROCEDURE_STATUS)
  {
    case 1  : traitement ;
              break ;
    case 2  : traitement ;
              break ;
        ...
    default : traitement ;
              break ;
  }
```

Le nombre de cas et la valeur du code correspondant dépendent de la définition qu'en a faite le concepteur du générateur. Il est alors nécessaire de se reporter à la documentation correspondante.

L'exemple suivant présente l'utilisation du code de retour de la procédure de calcul W(VEG,L,IBIAS) de la paire différentielle lors de son appel par la procédure VP1 d'un OTA. W(VEG,L,IBIAS) calcule la valeur W en fonction de la tension VEG, du courant IBIAS et de la longueur L. Deux cas de non validité des calculs peuvent se présenter, chacun correspondant à un code particulier : la largeur W demandée est trop grande pour la largeur maximale imposée par le procédé de fabrication, ou la largeur est trop petite. La procédure VP1 doit donc traiter ces deux cas rendus par W(VEG,L,IBIAS) et rendre le

résultat au niveau supérieur (par l'intermédiaire de codes de retour de procédure définis dans la procédure VP1).

La fonction de création de la paire différentielle (limitée à la fonction W(VEG,L,IBIAS)) où l'on voit la déclaration des paramètres, des codes de retour et des messages associés est :

```
CAIRO_BEGIN_CREATE(DP_MOS,char *trName1,char *trName2,char type,bool bulkToSource)

 ...

 // -------------------------
 // Déclaration des paramètres
 // -------------------------

 // Paramètres entrées
 CAIRO_DECLARE_PARAM("TEMP" ) ;
 CAIRO_DECLARE_PARAM("VDS"  ) ;  CAIRO_DECLARE_PARAM("VBS"  ) ;
 CAIRO_DECLARE_PARAM("VTH"  ) ;  CAIRO_DECLARE_PARAM("VDSAT") ;  CAIRO_DECLARE_PARAM("GMB"  ) ;
 CAIRO_DECLARE_PARAM("CGD"  ) ;  CAIRO_DECLARE_PARAM("CGS"  ) ;  CAIRO_DECLARE_PARAM("CGB"  ) ;
 CAIRO_DECLARE_PARAM("CBD"  ) ;  CAIRO_DECLARE_PARAM("CSD"  ) ;  CAIRO_DECLARE_PARAM("CM"   ) ;
 CAIRO_DECLARE_PARAM("CMB"  ) ;  CAIRO_DECLARE_PARAM("CMX"  ) ;  CAIRO_DECLARE_PARAM("CBS"  ) ;

 CAIRO_DECLARE_PARAM("VGS"  ) ;  CAIRO_DECLARE_PARAM("IBIAS") ;  CAIRO_DECLARE_PARAM("VEG"  ) ;
 CAIRO_DECLARE_PARAM("GDS"  ) ;  CAIRO_DECLARE_PARAM("GM"   ) ;

 CAIRO_DECLARE_PARAM("W"    ) ;  CAIRO_DECLARE_PARAM("L"    ) ;

 ...

 CAIRO_DECLARE_PROCEDURE("W(VEG,L,IBIAS)",CP_UNCOMPLETE,"TEMP",CP_FIXED,"VDS",CP_FIXED," // Entrees
                                                 "VBS",CP_FIXED,"VEG",CP_FIXED,     // Entrees
                                                 "L",CP_FIXED,                      // Entrees
                                                 "IBIAS",CP_NO,                     // Sorties
                                                 "W",CP_NO,"VDSAT",CP_NO,           // Sorties
                                                 "VTH",CP_NO,"VGS",CP_NO,           // Sorties
                                                 "GM",CP_NO,"GDS",CP_NO,            // Sorties
                                                 "GMB",CP_NO,"CGS",CP_NO,           // Sorties
                                                 "CGB",CP_NO,"CGD",CP_NO,           // Sorties
                                                 "CBD",CP_NO,"CSD",CP_NO,           // Sorties
                                                 "CM",CP_NO,"CMB",CP_NO,            // Sorties
                                                 "CMX",CP_NO,"CBS",CP_NO,           // Sorties
                                                 "MODE",CP_NO) ;                    // Sorties

 ...

 // ------------------------
 // Déclaration des messages
 // ------------------------

 CAIRO_DECLARE_PROCEDURE_MESSAGE(10,"W(VEG,L,IBIAS) : W too large") ;
 CAIRO_DECLARE_PROCEDURE_MESSAGE(11,"W(VEG,L,IBIAS) : W too small") ;

CAIRO_END_CREATE(DP_MOS)
```

Le code de la procédure W(VEG,L,IBIAS) montrant les différents cas de retour est :

```
 // ---------------------------
 // ------------ W(VEG,L,IBIAS)
 // ---------------------------

 CAIRO_BEGIN_PROCEDURE("W(VEG,L,IBIAS)")
```

```
// Récupération de TEMP, VDS et VBS

CAIRO_GET_VALUE("TEMP",temp)

CAIRO_GET_VALUE("VDS",vds)
CAIRO_GET_VALUE("VBS",vbs)

...

 // calcul de l, vdsat, vth, vgs
 mTr1 = CAIRO_GET_LOTRSINFO_M(trName) ;

 vth = CAIRO_MOS_VTH(trName,temp,l,(l + Wmin) * mTr1,0.0,vds,vbs) ;

 ids = ibias / 2.0 ;

 do
   {
     vgs = veg + vth ;
     w = CAIRO_MOS_W_VGS(trName,temp,l,ids,vgs,vds,vbs) ;

     if(cairo_mos_err == ERR_DIM)
      {
        failed = true ;

         switch(cairo_mos_reason)
           {
             case CAL_LARGE : CAIRO_SET_VALUE("IBIAS",ibias) ;
                              CAIRO_SET_RETURN_PROCEDURE(10) ;
                              break ;
             case CAL_SMALL : CAIRO_SET_VALUE("IBIAS",ibias) ;
                              CAIRO_SET_RETURN_PROCEDURE(11) ;
                              break ;
             default        : CAIRO_ERROR_MESSAGE("Unknown MOS error in %s\n",LOCATION.c_str()) ;
                              break ;
           }

        CAIRO_SET_VALUE("W",w) ;
        break ;
      }

     vdsat = CAIRO_MOS_VDSAT(trName,temp,l,w,vgs,vds,vbs) ;
     vth = CAIRO_MOS_VTH(trName,temp,l,w,vgs,vds,vbs) ;
     vegn = vgs - vth ;
   } while((fabs(veg - vegn) > EPSILON) && (failed == false)) ;

    ...

    CAIRO_SET_VALUE("W"    ,w    ) ;
    CAIRO_SET_VALUE("VDSAT",vdsat) ;
    CAIRO_SET_VALUE("VTH"  ,vth  ) ;
    CAIRO_SET_VALUE("VGS"  ,vgs  ) ;
    CAIRO_SET_VALUE("MODE" ,mode ) ;

    ...

 if(failed == false)
   {
     CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;
   }

END_PROCEDURE
```

Le module OTA définit lui-même deux codes de retour pour l'appel de la procédure VP1 (codes en relation avec le résultat de l'appel de la procédure W(VEG,L,IBIAS) de la

paire différentielle par la procédure VP1) :

```
CAIRO_BEGIN_CREATE(OTA,char *name,char type)

  ...

  // Messages de VP1
  CAIRO_DECLARE_PROCEDURE_MESSAGE(1 ,"VP1: DP: IBIAS is too large, try to decrease current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(2 ,"VP1: DP: IBIAS is too small, try to increase current") ;

  ...

CAIRO_END_CREATE(OTA)
```

Enfin, l'utilisation de la procédure W(VEG,L,IBIAS) dans le module OTA, avec le traitement des deux cas de retour, est donné ci-dessous. On peut noter que les messages associés aux cas d'échecs de la procédure VP1 sont différents de ceux associés à la procédure W(VEG,L,IBIAS), ce qui correspond à une interprétation, par la procédure VP1, de l'information renvoyée par la procédure W(VEG,L,IBIAS).

```
  // -----------------
  // ------------- VP1
  // -----------------

  CAIRO_BEGIN_PROCEDURE("VP1")

    ...

    do
      {
        va = va_next ;

        vds_dp = vout - va ;

        CAIRO_SET_PARAM_VAL("DP","VDS",vds_dp) ;

        CAIRO_COMPUTE("DP","W(VEG,L,IBIAS)") ;

        ON_PROCEDURE_STATUS
          switch(PROCEDURE_STATUS)
            {
              case 10 : CAIRO_SET_RETURN_PROCEDURE(1) ;
                        CAIRO_SET_VALUE("IBIAS",ibias_in) ;
                        CAIRO_SET_VALUE("VP1",vp1) ;
                        bad_return_vp1 = true ;
                        break ;
              case 11 : CAIRO_SET_RETURN_PROCEDURE(2) ;
                        CAIRO_SET_VALUE("IBIAS",ibias_in) ;
                        CAIRO_SET_VALUE("VP1",vp1) ;
                        bad_return_vp1 = true ;
                        break ;
              default : CAIRO_ERROR_MESSAGE("Unknown status %li for DP:W(VEG,L,IBIAS) in VP1\n",
                                            PROCEDURE_STATUS) ;
                        break ;
            }
        END_ON_PROCEDURE_STATUS

        CAIRO_GET_PARAM("DP","VTH",vth_dp) ;

        va_next = vin - vth_dp - veg_dp ;
      } while((fabs(va_next - va) > epsilon) && (bad_return_vp1 == false)) ;
```

```
   ...

   if(bad_return_vp1 != true)
     {
       CAIRO_SET_VALUE("VP1",vp1) ;
       CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;
     }

 END_PROCEDURE
```

Erreur fatale dans une procédure

Certains cas imposent un traitement fatal de l'échec. Le concepteur a à sa disposition la fonction :

```
FATAL_ERROR_PROCEDURE(string name,const char *message,string location) ;
```

name désigne le nom de la procédure fautive (on peut utiliser PROCEDURE_NAME)
message est le message associé
location est l'endroit où l'erreur s'est produite.

Il s'agit d'erreurs fatales qui arrêtent forcément l'exécution.

8.4 Formation d'un exécutable

Les générateurs CAIRO+ ne sont que des fonctions au sens C du terme. Il faut donc écrire un programme principal qui se chargera de créer le circuit désiré et mènera à bien l'exploration du domaine de conception en prenant en compte les spécifications fonctionnelles définies par le concepteur.

Le programme principal est une séquence d'appels aux fonctions de création et de dimensionnement qui doit comporter dans l'ordre :

```
int main(int argc,char *argv[])
{
  CAIRO_NEW_CHIP               ----> création du circuit

    CAIRO_CREATE               ----> création du module principal

    Positionnement éventuel de contraintes

    -----
    | CAIRO_SET_PARAM          |---\ DSES
    |
    | CAIRO_COMPUTE            |---> appel de la procédure de dimensionnement
    |
    | Traitement éventuel
    | du code de retour de la procédure
    |
    | CAIRO_GET_PARAM          |---/
    -----

    CAIRO_RESHAPE              ----> Choix du facteur de forme du circuit

  CAIRO_SAVE_CHIP

  return 0 ;
}
```

La fonction main commence par un appel à CAIRO_NEW_CHIP dont le prototype est :

```
CAIRO_NEW_CHIP(char *chip_name)
```

chip_name est le nom du circuit (différent du nom du module racine)

Puis vient la fonction de création, suivie de l'appel à la fonction de dimensionnement du DSES. Il faut bien sûr gérer le code de retour de la procédure. Enfin, le placement et le routage sont effectués lors de l'appel à CAIRO_RESHAPE. C'est à ce moment que la fonction CAIRO_BEGIN_LAYOUT est implicitement appelée (figure 8.8).

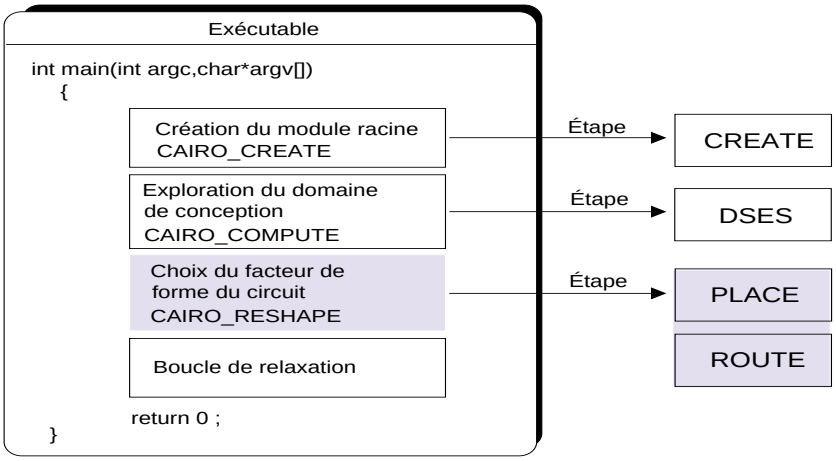


FIG. 8.8: Le placement et le routage sont déclenchés par la fonction CAIRO_RESHAPE

La fonction CAIRO_RESHAPE peut prendre trois formes différentes suivant la contrainte physique à appliquer au module racine :

```
CAIRO_RESHAPE()
CAIRO_RESHAPE_WIDTH(double width)
CAIRO_RESHAPE_HEIGHT(double width)
```

La première forme donne un circuit possédant un facteur de forme carré ou se rapprochant du carré. La deuxième forme permet d'imposer une contrainte horizontale, la troisième forme permet d'imposer une contrainte verticale.

La sauvegarde des fichiers est assurée par la fonction CAIRO_SAVE_CHIP :

```
CAIRO_SAVE_CHIP(char *chip_name)
```

chip_name est le nom du circuit (différent du nom du module racine)

L'exemple suivant montre l'appel de la procédure VP1 d'un OTA, procédure calculant le point de polarisation en fonction du courant et de diverses tensions :

```
int main(int argc, char *argv[])
{
  bool   dummy     = true       ;
  bool   ring      = true       ;

  double temp      = 300.15     ;

  double vin       = 0.60       ;
  double vout      = 0.60       ;
  double ibias     = 500.0e-6   ;
  double ibias_cal = 0.0        ;
  double vp1       = 0.0        ;
  double cload     = 5.0e-12    ;
  double cload_cal = 5.0e-12    ;
  double veg_bias  = 0.2        ;
  double veg_dp    = 0.2        ;

  CAIRO_NEW_CHIP("testOTA") ;

  // --------
  // Création
  // --------

  CAIRO_CREATE("libOTA","OTA","OTA",TRANSP,ring,dummy) ;

  // ----
  // DSES
  // ----

  // ***** Calcul du point de polarisation

  CAIRO_SET_PARAM_VAL("OTA","TEMP" ,temp ) ;

  CAIRO_SET_PARAM_VAL("OTA","VIN"     ,vin     ) ;
  CAIRO_SET_PARAM_VAL("OTA","VOUT"    ,vout    ) ;
  CAIRO_SET_PARAM_VAL("OTA","VEG_BIAS",veg_bias) ;
  CAIRO_SET_PARAM_VAL("OTA","VEG_DP"  ,veg_dp  ) ;

 // ----- Appel de VP1

  CAIRO_COMPUTE("OTA","VP1") ;

  // ----- Récupération de vp1
  CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;

  CAIRO_RESHAPE() ;

  CAIRO_SAVE_CHIP("testOTA") ;

  exit(0) ;
}
```

L'activation de la boucle de relaxation doit être demandée dans la fonction main. Le concepteur utilise la variable booléenne CAIRO_SYSTEM_IS_STABLE de la manière suivante :

```
int main(int argc, char *argv[])
{
  double temp      = 300.15     ;

  double vin       = 0.60       ;
  double vout      = 0.60       ;
```

```
 double ibias     = 530.0e-6   ;
 double ibias_cal = 0.0        ;
 double vp1       = 0.0        ;

 double cload_in  = 5.0e-12    ;
 double cload_max = 5.0e-12    ;
 double cload_out = 0.0        ;
 double ugf       = 300.0e+06  ;
 double ugf_cal   = 0.0        ;
 double pm_low    = 73.0       ;
 double pm_high   = 76.0       ;
 double pm_cal    = 0.0        ;
 double gbw_cal   = 0.0        ;
 double cof_cal   = 0.0        ;
 double sr_cal    = 0.0        ;
 double gain_cal  = 0.0        ;
 double veg_bias  = 0.10       ;
 double veg_dp    = 0.10       ;
 double l_init    = 4.2e-6     ;
 double l_step    = 0.0        ;

 CAIRO_NEW_CHIP("testOTA") ;

 // --------
 // Création
 // --------

 CAIRO_CREATE("libOTA","OTA","OTA",TRANSP) ;

 // ----
 // DSES
 // ----

 // ***** Calcul du point de polarisation

 CAIRO_SET_PARAM_VAL("OTA","TEMP" ,temp ) ;

 CAIRO_SET_PARAM_VAL("OTA","VIN"     ,vin     ) ;
 CAIRO_SET_PARAM_VAL("OTA","VOUT"    ,vout    ) ;
 CAIRO_SET_PARAM_VAL("OTA","VEG_BIAS",veg_bias) ;
 CAIRO_SET_PARAM_VAL("OTA","VEG_DP"  ,veg_dp  ) ;

// ----- Appel de VP1
 CAIRO_COMPUTE("OTA","VP1") ;

 // ----- Récupération de vp1
 CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;

 // ***** Calcul du UGF

 CAIRO_SET_PARAM("OTA","CLOAD",cload_in) ;

 // Contraintes pour ugf et pm
 CAIRO_SET_PARAM_INF("OTA","UGF",ugf) ;
 CAIRO_SET_PARAM_INF("OTA","PM",pm_low) ;
 CAIRO_SET_PARAM_SUP("OTA","PM",pm_high) ;

 while(CAIRO_SYSTEM_IS_STABLE == false)
   {
     CAIRO_COMPUTE("OTA","UGF") ;

     CAIRO_RESHAPE() ;
   }
```

```
  CAIRO_GET_PARAM("OTA","UGF",ugf) ;
  CAIRO_SAVE_CHIP("chip") ;

  return 0 ;
}
```

8.5 Résumé

Ce chapitre a présenté les principaux services fournis par le langage CAIRO+ au concepteur analogicien, en insistant sur les fonctions permettant d'écrire un nouveau générateur de module. Le manuel d'utilisation peut être consulté pour plus de détails, le chapitre 9 donnant un exemple complet reposant sur la réalisation d'un OTA.

Chapitre 9

Exemples et résultats

Sommaire

9.1 Conception hiérarchique d'un OTA 96
9.1.1 Présentation d'un OTA 96
9.1.2 Choix des dispositifs élémentaires 98
9.1.3 Équations de l'OTA 100
9.1.4 Dimensionnement électrique de l'OTA 101
9.1.5 Procédure $VP1$ 103
9.1.6 Présentation du code réalisant l'OTA 107
9.1.7 Simulations et résultats 110
9.2 Conception hiérarchique d'un filtre passe-bas pour un modulateur $\Sigma\Delta$ à temps continu et mode courant 119
9.2.1 Objectif et architecture 119
9.2.2 Hiérarchie d'un module « Filtre passe-bas » 121
9.2.3 Dimensionnement électrique 122
9.2.4 Simulations et résultats 126
9.3 Conclusion 135

Le langage CAIRO+ a été conçu pour permettre de réaliser aisément des générateurs de circuits analogiques. La méthodologie employée et décrite dans les précédents chapitres autorise la capitalisation des connaissances sous une forme hiérarchique. CAIRO+ apporte une réponse au problème de changement de spécification et à celui de migration technologique qui se posent régulièrement dans le contexte de la conception de circuits intégrés analogiques. L'investissement consacré à la conception de générateurs de circuits se traduit par un gain de temps lors de la réutilisation, dans telle ou telle application, des composants ainsi créés. La plate-forme CAIRO+ est déjà utilisée au sein de la spécialité ACSI (Architecture et Conception des Systèmes Intégrés) de la mention informatique du master de l'UPMC.

Ce chapitre présente deux exemples de conception de générateurs avec le langage CAIRO+. Le premier est un générateur d'amplificateur opérationnel à transconductance. Il s'agit d'un module de premier niveau (le code source est donné en annexe D). Le second est un générateur de filtre passe-bas utilisé dans un modulateur $\Sigma\Delta$ à temps continu et mode courant. Il s'agit d'un module de deuxième niveau (le code source est donné en annexes F et H). Dans ces deux cas la conception du générateur s'appuie sur la connaissance du concepteur, telle qu'elle peut apparaître dans la littérature.

9.1 Conception hiérarchique d'un OTA

9.1.1 Présentation d'un OTA

Un amplificateur différentiel à transconductance est un générateur de courant commandé par une tension différentielle : le courant de sortie est proportionnel à la tension d'entrée différentielle multipliée par la transconductance de l'amplificateur (figure 9.1).

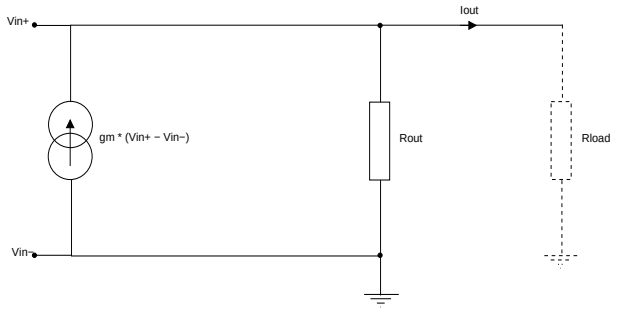


FIG. 9.1: Amplificateur différentiel à transconductance

On appelle couramment OTA (*Operational Transconductance Amplifier*) un amplificateur de tension obtenu à partir d'un amplificateur à transconductance à entrée différentielle fonctionnant avec la sortie en circuit ouvert, ayant pour seule résistance de charge sa propre résistance de sortie (figure 9.2).

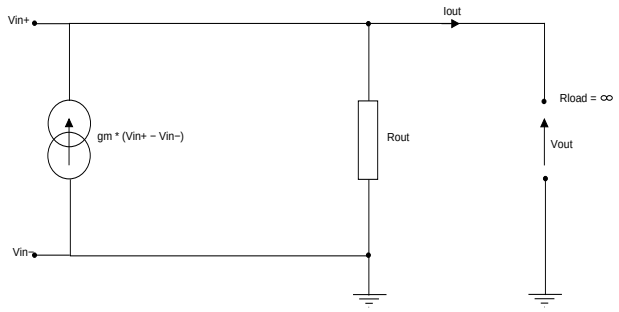


FIG. 9.2: OTA à transconductance classique

Dans la mesure où la résistance de sortie d'un amplificateur à transconductance est très élevée (idéalement infinie), on obtient un gain en tension très grand :

$$A_V = g_m R_{out} \tag{9.1}$$

En micro-électronique, un amplificateur différentiel est réalisé à partir d'une cellule appelée « paire de transistors différentielle ». En technologie CMOS, il s'agit de transistors MOS. Le schéma de principe d'un amplificateur différentiel à transconductance est donné figure 9.3.

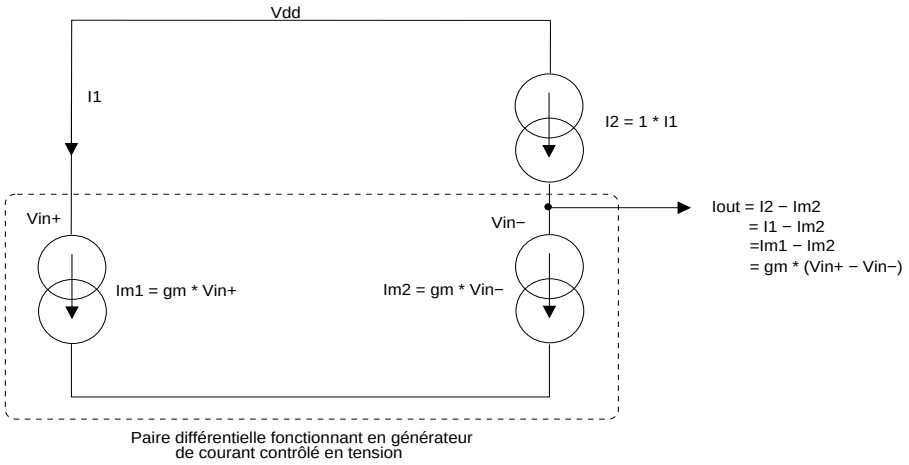


FIG. 9.3: Schéma de principe de l'amplification

Sachant que, sous certaines conditions, un transistor MOS fonctionne comme une source de courant commandée par la tension de grille (figure 9.4), un amplificateur différentiel à

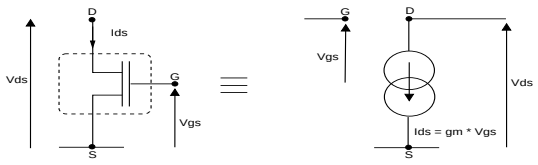


FIG. 9.4: Transistor MOS et son modèle simplifié

transconductance fonctionne comme une « balance » qui ne réagit qu'à la différence entre les tensions d'entrée.

Ce fonctionnement « idéal » n'est valable que sous certaines conditions. Premièrement, puisqu'il s'agit d'une « balance », les deux branches doivent être parfaitement équilibrées. Donc, les deux transistors doivent avoir un fonctionnement identique : ils doivent avoir des dimensions, des géométries ainsi que des conditions de fonctionnement identiques, notamment en ce qui concerne la température et la polarisation électrique.

Deuxièmement, pour qu'un transistor se comporte comme une source de courant commandée (quasi)linéairement par une tension, il doit être correctement polarisé. Cela veut dire qu'au repos un courant continu (DC) ou une tension continue grille-source doivent lui être imposés.

Pour mettre en œuvre un amplificateur différentiel à transconductance, on utilise le schéma donné figure 9.6. On appelle « régime DC », ou « régime de repos », le régime dans lequel la différence entre les tensions d'entrée est nulle — mais pas les tensions d'entrée elles-mêmes. En effet, par symétrie du circuit, le courant de sortie est alors nul.

Dans le schéma de la figure 9.6, le transistor N5 fonctionne en régime d'une source de courant continue. Les deux transistors P3 et P4 réalisent une source de courant commandée par un courant avec un gain de 1. Cette cellule, très utilisée en micro-électronique, est appelée « miroir de courant », puisque le courant d'un transistor est recopié par l'autre. Pour préserver la symétrie du circuit, le gain doit être strictement unitaire ; pour cette raison, on impose à ces transistors les mêmes contraintes d'appariement qu'aux transistors de la paire différentielle.

Le transistor N5 génère un courant continu de polarisation I_{bias} qui, grâce à la symétrie du circuit, se divise en deux en assurant une polarisation identique aux deux branches. Également dans un souci de polarisation, on applique aux deux entrées une tension continue (DC) identique.

9.1.2 Choix des dispositifs élémentaires

Les dispositifs élémentaires utilisés sont naturellement le dispositif élémentaire transistor simple (pour BIAS), le dispositif élémentaire paire différentielle (pour DP) et le dispositif élémentaire miroir de courant (pour CM), tous disponibles dans la bibliothèque *libMOS* fournie par CAIRO+. L'OTA, ne faisant appel qu'aux dispositifs élémentaires, est un module de premier niveau.

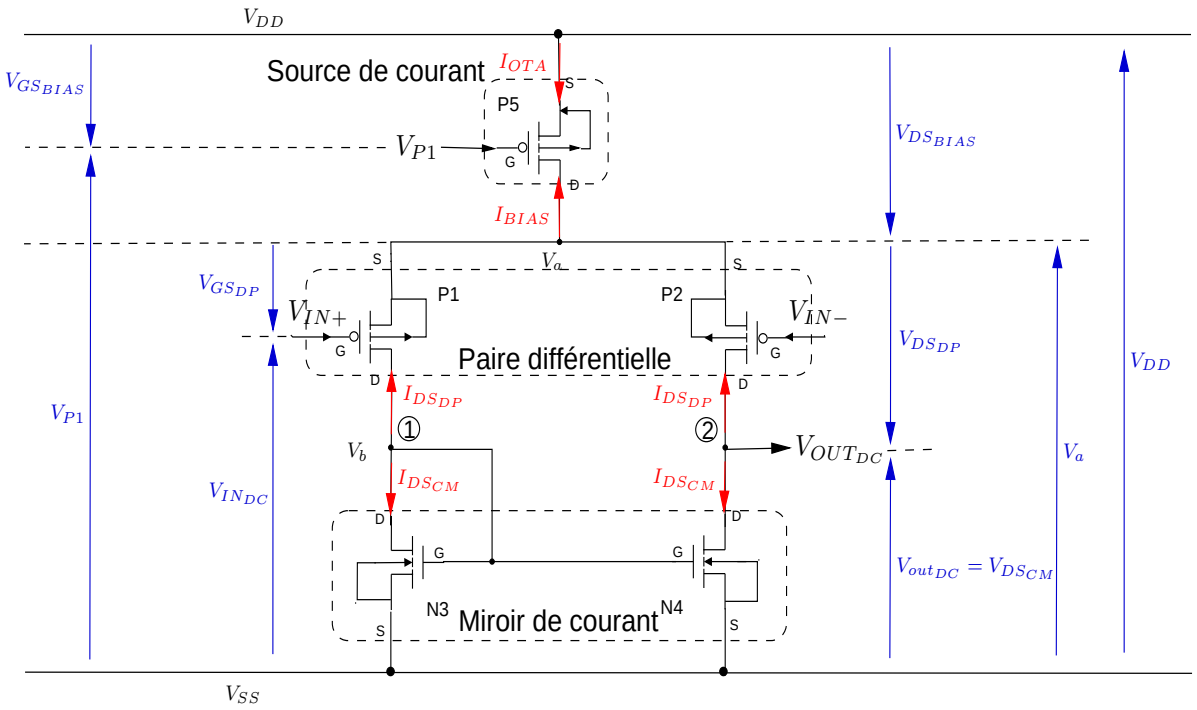


FIG. 9.5: OTA de type P - Tensions et courants associés

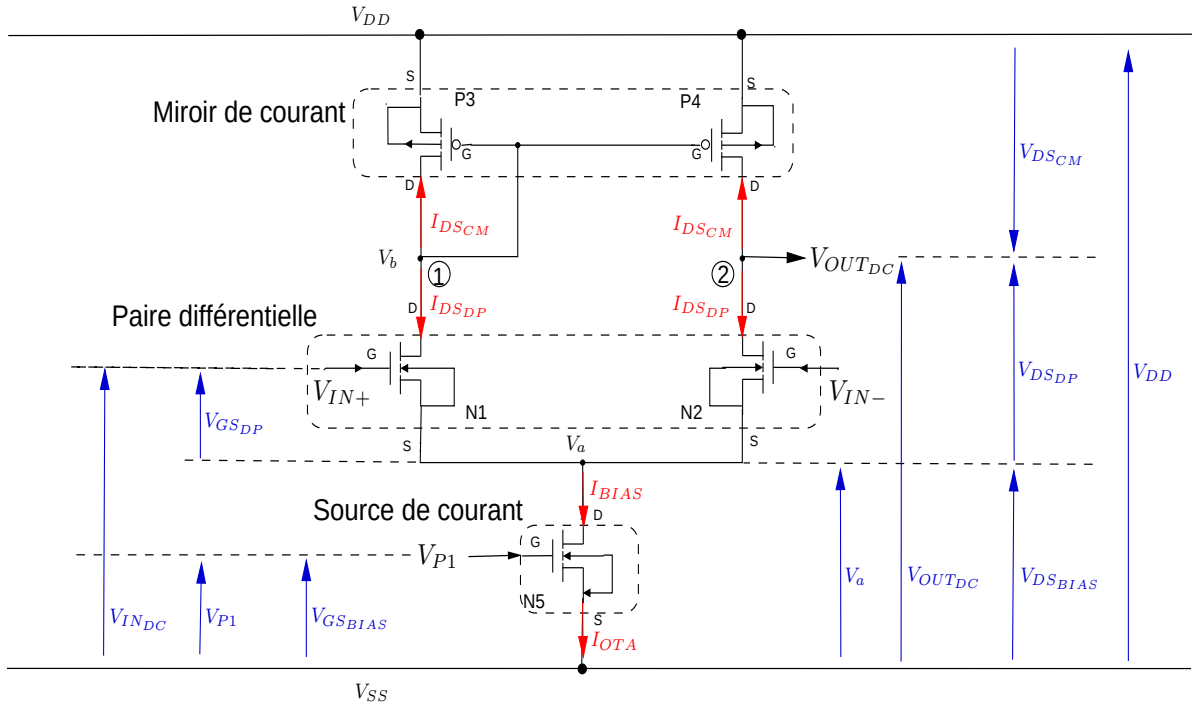


FIG. 9.6: OTA de type N - Tensions et courants associés

9.1.3 Équations de l'OTA

La table ci-dessous résume les notations utilisées dans les équations de l'OTA.

$UGF\ (Hz)$	la fréquence de transition (*Unity Gain Frequency*)
$GBW\ (Hz)$	le produit gain-bande (*Gain Bandwidth*)
$PM\ (degré)$	la marge de phase
$COF\ (Hz)$	la fréquence de coupure (*Cut-Off Frequency*)
$SR\ (volt/s)$	la vitesse maximale de variation de la tension de sortie (*Slewrate*)
$Ad(j\omega)$	l'expression complexe de la fonction de transfert
$\|Ad(\omega)\|$	l'évolution du gain en fonction de la pulsation
Ad_0	le gain DC
$\omega_T\ (rad/s)$	la pulsation au point de transition (gain unitaire)
$\omega_0\ (rad/s)$	la pulsation du zéro de la fonction de transfert
$\omega_1\ (rad/s)$	la pulsation du second pôle de la fonction de transfert
$\omega_2\ (rad/s)$	la pulsation du premier pôle de la fonction de transfert
C_1 *et* $C_2\ (farad)$	les capacités parasites aux nœuds ① et ② (figures 9.5 et 9.6)
G_1 *et* $G_2\ (siemens)$	les conductances aux nœuds ① et ② (figures 9.5 et 9.6)
$C_{load}\ (farad)$	la capacité de charge en V_{OUT}
$C_{XY_{ZZ}}\ (farad)$	la capacité parasite entre les nœuds X et Y du dispositif élémentaire ZZ
$g_{m_{ZZ}}\ (siemens)$	la transconductance du dispositif élémentaire ZZ

Les performances de l'OTA prises en compte par les procédures de dimensionnement se modélisent par les équations suivantes (pour le modèle considéré, d'après [Por05]) :

$$UGF = \frac{\omega_T}{2 \cdot \pi} \tag{9.2}$$

$$COF = \frac{\omega_2}{2 \cdot \pi} \tag{9.3}$$

$$GBW = Ad_0 \cdot COF \tag{9.4}$$

$$PM = 180 + \left(\arctan \frac{\omega_T}{\omega_0} - \arctan \frac{\omega_T}{\omega_1} - \arctan \frac{\omega_T}{\omega_2} \right) \cdot \frac{180}{\pi} \tag{9.5}$$

$$SR = \frac{I_{BIAS}}{C_2} \tag{9.6}$$

avec

$$|Ad|_{\omega=\omega_T} = 1 \tag{9.7}$$

$$|Ad| = Ad_0 \cdot \frac{\sqrt{1 + \left(\frac{\omega}{\omega_0}\right)^2}}{\sqrt{1 + \left(\frac{\omega}{\omega_1}\right)^2} \cdot \sqrt{1 + \left(\frac{\omega}{\omega_2}\right)^2}} \tag{9.8}$$

$$Ad_0 = \frac{1}{2} \cdot \frac{gm_{DP} \cdot (gm_{CM} + G1)}{G1 \cdot G_2} \tag{9.9}$$

$$\omega_0 = \omega_1 \cdot \left(1 + \frac{gm_{CM}}{G1}\right) \tag{9.10}$$

$$\omega_1 = \frac{G1}{C_1} \approx \frac{g_{m_{CM}}}{C_1} \tag{9.11}$$

$$\omega_2 = \frac{gds_{DP} + gds_{CM}}{C_2} \tag{9.12}$$

$$C_1 = C_{GD_{DP}} + C_{BD_{DP}} + 2C_{GS_{CM}} + 2C_{GB_{CM}} + C_{BD_{CM}}$$

$$C_2 = C_{GD_{DP}} + C_{BD_{DP}} + C_{GD_{CM}} + C_{BD_{CM}} + C_{load} \tag{9.13}$$

$$G_1 = gds_{DP} + gds_{CM} + gm_{CM} \tag{9.14}$$

$$G_2 = gds_{DP} + gds_{CM} \tag{9.15}$$

9.1.4 Dimensionnement électrique de l'OTA

Le choix des paramètres d'entrée-sortie de la procédure de dimensionnement de l'OTA dépend à la fois des équations modélisant cet OTA (équations 9.2 à 9.6) et des fonctions disponibles au niveau du DSES des dispositifs élémentaires utilisés. Les procédures de dimensionnement utilisées reprennent les méthodes proposées par Jacky Porte dans [Por05] et par Mohamed Dessouky dans [Des01].

Le dimensionnement d'un circuit consiste à déterminer des paramètres inconnus du circuit connaissant ses performances souhaitées. Un dimensionnement est donc une opération inverse, du moins en partie, à celle d'analyse, où, à partir de tous les paramètres du circuit, par simulation ou par calcul, on détermine les performances.

L'analyse de la plupart des circuits de type amplificateur consiste essentiellement en deux étapes (figure 9.7). L'analyse en régime de repos (ou analyse DC, calcul de la polarisation) calcule tous les courants et tensions inconnus du circuit, connaissant les dimensions et les tensions des sources, par exemple, les tensions V_P, V_{IN}, V_{DD} et V_{SS} du schéma de la figure 9.6. L'analyse du schéma équivalent petit signal consiste ensuite à déterminer la fonction de transfert pour le régime DC préalablement déterminé. L'analyse des performances telles que le bruit, la dynamique d'entrée-sortie, la sensibilité, la stabilité, etc. se réduit le plus souvent à l'une de ces deux analyses. En tenant compte de ces principes, le

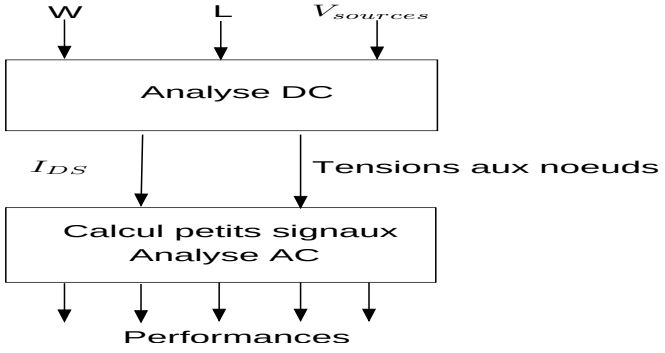


FIG. 9.7: Les deux étapes d'analyse d'un amplificateur

dimensionnement de l'OTA est réalisé par deux procédures : VP1 et UGF (tableau 9.1).

TAB. 9.1: Procédures de l'OTA

Paramètres imposés en entrée		Paramètres calculés	Nom	Classe
TEMP	IBIAS VIN VOUT VEG_BIAS VEG_DP L	VP1 W	VP1	CP_UNCOMPLETE
	(IBIAS) (VEG_BIAS) (VEG_DP) CLOAD	UGF PM IBIAS_OUT SR GAIN COF GBW (VP1)	UGF	CP_COMPLETE

9.1.5 Procédure $VP1$

La procédure VP1 effectue les calculs liés au régime DC. L'inversion complète de l'opération d'analyse DC, *i.e.* détermination de toutes les dimensions à partir de toutes les tensions et courants DC du circuit n'est pas possible pour plusieurs raisons. Premièrement, le nombre de paramètres à déterminer,(*i.e.* paramètres dimensionnels et tensions de sources DC), n'est pas forcément égal au nombre de tensions et de courants DC connus du circuit. Même si cela était le cas, il n'est pas possible, dans le cas général, d'établir une relation explicite entre les deux ensembles. Deuxièmement, ceci n'est pas nécessaire, car le concepteur connaît, dans beaucoup de cas, certaines dimensions caractéristiques de son circuit et *quelques* paramètres DC caractéristiques. Il souhaite alors en déduire les paramètres manquants, aussi bien au niveau des dimensions et paramètres des sources qu'au niveau des tensions et courants inconnus du circuit.

Ainsi, cette procédure prend en argument le courant DC (I_{BIAS}), les tensions DC d'entrée et de sortie, les tensions effectives de grille de la paire différentielle et de la source de courant (transistor N5) et les longueurs de tous les transistors (L_i).

La procédure VP1 retourne les largeurs de tous les transistors (W_i) et calcule la tension de polarisation à appliquer sur la grille du transistor N5 (appelée justement V_{P1}).

On peut dire que la procédure VP1 résout les équations algébriques non-linéaires obtenues à partir des équations de Kirchoff pour le circuit de la figure 9.6. En général, une méthode numérique doit être utilisée. Cependant, au vu du nombre important de paramètres du circuit connus, un algorithme de dimensionnement simplifié peut être mis en place.

Il est facile de voir que, si l'on connaît la tension V_a (figure 9.5 ou 9.6), tous les paramètres inconnus peuvent être déterminés un par un en appelant les procédures incomplètes des dispositifs élémentaires. Malheureusement, cette tension est liée d'une manière non-linéaire avec d'autres paramètres inconnus du circuit (par exemple, avec la tension de seuil (V_{th}) du transistor N5 ou de la paire différentielle).

Si l'on prend pour deuxième inconnue la tension de seuil du transistor N5, à partir du circuit de la figure 9.6 on a :

$$\begin{aligned} V_a &= V_{IN} - V_{GS_{DP}} && (9.16)\\ &= V_{IN} - V_{TH_{DP}} - V_{EG_{DP}} && (9.17) \end{aligned}$$

et

$$V_{DS_{DP}} = V_{OUT} - V_a. \quad (9.18)$$

Par ailleurs, du modèle électrique du transistor, il est possible de déduire que :

$$V_{TH_{DP}} = V_{TH}(L, I_{DS_{DP}}, V_{EG_{DP}}, V_{DS_{DP}}, V_{BS_{DP}}, Temp), \quad (9.19)$$

où $V_{TH}(...)$ est une procédure incomplète du dispositif élémentaire « paire différentielle ».

Ces trois dernières équations non-linéaires doivent être résolues numériquement : c'est ce que fait la procédure VP1 en utilisant une boucle.

Une fois V_a et $V_{TH_{DP}}$ connus :

- On appelle la procédure $W(L, I_{DS}, V_{EG}, V_{DS}, V_{BS}, Temp)$ pour les dispositifs N5 et paire différentielle. Cette procédure retourne également la tension de seuil – V_{bias} est alors connu ;
- On appelle la procédure $W(L, I_{DS}, V_{GS}, V_{DS}, V_{BS}, Temp)$ pour le dispositif élémentaire « miroir de courant ».

Gestion des échecs de la procédure $VP1$

La boucle de dimensionnement ne fait pas apparaître la gestion des échecs rendus par les procédures des dispositifs élémentaires. Or, les procédures des dispositifs élémentaires appelées par $VP1$ peuvent indiquer *via* leur code de retour que :

- la largeur obtenue W est trop grande pour la technologie cible et les conditions de polarisation
- la largeur obtenue W est trop petite pour la technologie cible et les conditions de polarisation

En outre, une vérification est faite pour contrôler que le régime de fonctionnement des transistors est le régime saturé (c'est un paramètre de retour des procédures des dispositifs élémentaires).

$VP1$ ne réalise aucune optimisation et ne peut agir sur ses paramètres en cas de problème de dimensionnement, cette tâche incombe à l'appelant. Cependant, la procédure $VP1$ peut effectuer un diagnostic des cas d'échec renvoyés par les dispositifs élémentaires. Notamment, elle considère qu'un dépassement de l'intervalle permis pour les largeurs peut être corrigé en modifiant la valeur du courant de polarisation ou la longueur des transistors. La procédure $VP1$ peut signaler, au niveau supérieur, la raison d'un échec en utilisant un code de retour, comme par exemple « IBIAS trop grand », « IBIAS trop petit ». Un tel échec peut se produire lors du dimensionnement de chacun des 3 dispositifs élémentaires mis en jeu dans l'OTA (soit 6 codes au total). Dans le cas où un des dispositifs élémentaires n'est pas en régime saturé, un code d'erreur est également retourné.

Calcul de la fréquence de transition : la procédure UGF

La procédure UGF est une inversion partielle de la deuxième étape de l'analyse du circuit : l'analyse petit signal. Elle s'appuie sur les équations qui relient les paramètres DC du circuit (les dimensions, les tensions et les courants de polarisation) avec les performances telles que la fréquence de transition, le gain, la marge de phase, etc. La procédure UGF fait appel à la procédure VP1 pour connaître le point de polarisation du circuit. Elle calcule ensuite les performances, les compare avec les spécifications et, selon l'algorithme défini par le concepteur, modifie les paramètres d'entrée de VP1 en procédant ainsi en boucle d'itération.

Les paramètres d'entrée de la procédure UGF sont les performances souhaitées (PM_{min}, PM_{max}, UGF_{min}, $CLOAD$) et aussi les paramètres DC du circuit imposés par le concepteurs ($V_{IN_{DC}}$, $V_{OUT_{DC}}$, tous les V_{EG}). Ces derniers paramètres sont retransmis tels quels lors de chaque appel de la procédure $VP1$. En revanche, les paramètres L_i et I_{bias} requis par la procédure $VP1$ servent de variables d'optimisation. Il est possible de donner des valeurs initiales à ces variables qui sont susceptibles d'être modifiées par l'algorithme de dimensionnement de la procédure UGF. La procédure de dimensionnement UGF peut être résumée ainsi :

1. Récupérer PM_{min}, PM_{max}, UGF_{min}, $CLOAD$
2. A l'étape d'initialisation, positionner tous les L et I_{bias} aux valeurs initiales et appeler VP1
3. Calculer les paramètres petits signaux nécessaires aux calculs de PM_{cal} et UGF_{cal} (équations 9.2 et 9.5)
4. Calculer PM_{cal} et UGF_{cal}
5. Boucle sur la marge de phase
   (a) Si $PM_{cal} < PM_{min}$
      i. Décrémenter la longueur de tous les transistors d'un pas de grille physique : $L = L - step$ (en vérifiant que $L > L_{min}$)
         A. Positionner les L en vue de l'appel de VP1
         B. Appeler VP1
         C. Aller en 3
      ii. Si $PM_{max} < PM_{cal}$
         A. Incrémenter la longueur de tous les transistors d'un pas de grille physique : $L = L + step$
         B. Positionner les L en vue de l'appel de VP1
         C. Appeler VP1
         D. Aller en 3
   (b) Boucle sur la fréquence de transition
      i. Si $UGF_{cal} < UGF_{min}$ ou $\frac{UGF_{min} - UGF_{cal}}{UGF_{cal}} > \epsilon$
         A. Augmenter la valeur du courant I_{BIAS} : $I_{BIAS} = I_{BIAS} \cdot \frac{UGF_{min}}{UGF_{cal}}$
         B. Positionner I_{BIAS} en vue de l'appel de VP1
         C. Positionner les longueurs de tous les transistors à leurs valeurs initiales
         D. Appeler VP1
         E. Aller en 3
   (c) Calculer le slew rate
   (d) Calculer le gain DC

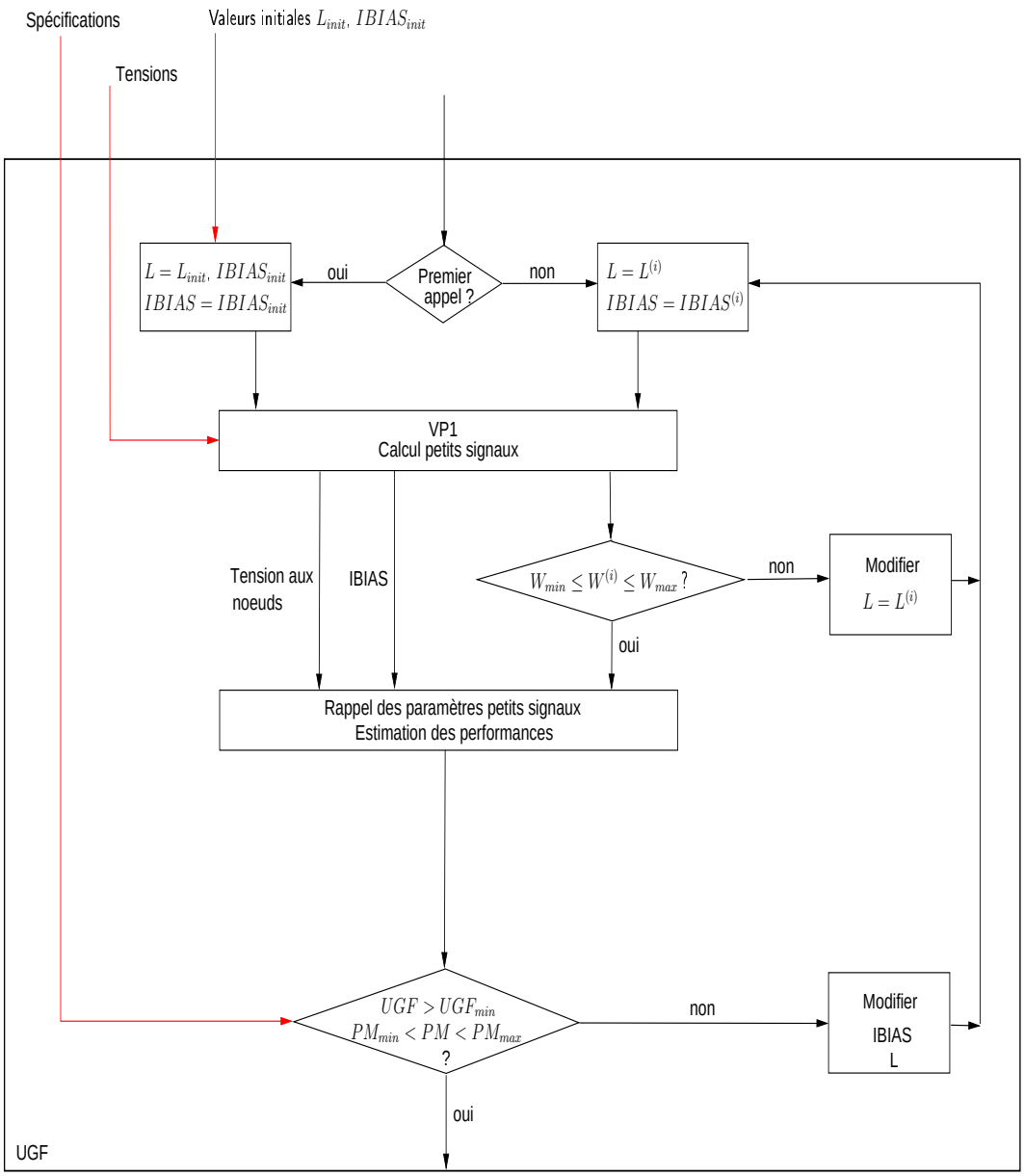


FIG. 9.8: Procédures UGF

Gestion des échecs de la procédure UGF

La procédure UGF ne fait pas directement appel aux procédures de dimensionnement du transistor, de la paire différentielle et du miroir de courant (mis à part les procédures $direct\_size$). En revanche, elle doit gérer les cas d'échec possibles de la procédure $VP1$ qu'elle utilise. Comme nous l'avons dit dans le paragraphe 9.1.5, l'échec le plus fréquent de cette procédure est un dépassement, par un des dispositifs élémentaires, de l'intervalle des largeurs autorisées. Pour résoudre ce problème, l'appelant de $VP1$ peut réagir de différentes manières : modifier L, VGS, VEG ou IDS. Ici, la procédure UGF réagit en

modifiant la longueur de tous les dispositifs élémentaires.

9.1.6 Présentation du code réalisant l'OTA

Ce paragraphe présente des extraits significatifs du code du générateur de l'OTA de type P, le code complet pouvant être consulté en annexe D.

Le concepteur du générateur OTA doit écrire trois fonctions : la fonction de création, la fonction d'exploration du domaine de conception et la fonction de routage.

Fonction de création

La fonction de création ne prend en argument, outre le nom de l'instance (argument obligatoire), que le type de l'OTA et le type des transistors utilisés. On y retrouve l'instanciation des trois dispositifs élémentaires (voir figure 8.3 page 77) :

```
CAIRO_BEGIN_CREATE(OTA,char *name,char type)

  // Transistor simple BIAS de type P
  CAIRO_CREATE("libMOS","TR_MOS","BIAS","P5",TRANSP,true) ;

  // Paire differentielle DP de type P
  CAIRO_CREATE("libMOS","DP_MOS","DP","P1","P2",TRANSP,true) ;

  // Miroir de courant simple CM de type N
  CAIRO_CREATE("libMOS","CM_MOS","CM","N3","N4",TRANSN,true) ;
```

Puis vient la déclaration des connecteurs :

```
  CAIRO_IO("VP1" ,CP_WEST ) ; // Tension de polarisation
  CAIRO_IO("VIN1",CP_WEST ) ; // Entree différentielle +
  CAIRO_IO("VIN2",CP_WEST ) ; // Entrée différentielle -
  CAIRO_IO("VOUT",CP_EAST ) ; // Sortie
  CAIRO_IO("VDD" ,CP_EAST ) ; // Alimentation
  CAIRO_IO("VSS" ,CP_SOUTH) ; // Alimentation
```

Les instances sont ensuite connectées (schéma électrique non dimensionné) et placées dans un conteneur vertical (voir figure 8.5 page 78) :

```
  // BIAS : drain gate source (bulk relié à la source en interne)
  CAIRO_CONNECT("BIAS","drain => VA","gate => VP1","source => VDD");

  // DP : drain1 drain2 gate1 gate2 source (bulk relié à la source en interne)
  CAIRO_CONNECT("DP","drain1 => VOUT","drain2 => VB","gate1 => VIN2","gate2 => VIN1","source => VA") ;

  // CM : drain1 drain2 source (bulk relié à la source en interne)
  CAIRO_CONNECT("CM","drain1 => VB","drain2 => VOUT","source => VSS");

  CAIRO_VERTICAL_CONTAINER(name,"CM"   ,NOSYM,3 * PITCH,2 * PITCH,SD_NWELL_PTIE,SD_NWELL_PTIE,
                               "DP"   ,NOSYM,2 * PITCH,SD_NTIE ,SD_NTIE      ,SD_NTIE      ,
                               "BIAS",NOSYM,2 * PITCH,SD_NTIE ,PITCH * 2    ,PITCH * 2     ) ;
```

Le schéma électrique non dimensionné étant décrit, le concepteur déclare l'ensemble des paramètres utilisés par les procédures de dimensionnement :

```
  CAIRO_DECLARE_PARAM("TEMP"    ) ;
  CAIRO_DECLARE_PARAM("VEG_BIAS") ;
  CAIRO_DECLARE_PARAM("VEG_DP"  ) ;
  CAIRO_DECLARE_PARAM("L_INIT"  ) ;
  CAIRO_DECLARE_PARAM("VIN"     ) ;
  CAIRO_DECLARE_PARAM("VOUT"    ) ;

  CAIRO_DECLARE_PARAM("VP1"      ) ;
  CAIRO_DECLARE_PARAM("UGF"      ) ;
  CAIRO_DECLARE_PARAM("PM"       ) ;
  CAIRO_DECLARE_PARAM("GBW"      ) ;
  CAIRO_DECLARE_PARAM("COF"      ) ;
  CAIRO_DECLARE_PARAM("SR"       ) ;
  CAIRO_DECLARE_PARAM("GAIN"     ) ;
  CAIRO_DECLARE_PARAM("CLOAD"    ) ;

  CAIRO_DECLARE_PARAM("IBIAS") ;
```

Les deux procédures du générateur, $VP1$ et UGF, sont déclarées :

```
CAIRO_DECLARE_PROCEDURE("VP1",CP_UNCOMPLETE,"TEMP",CP_FIXED,"VEG_BIAS",CP_FIXED,"VEG_DP",CP_FIXED,
                        "L_INIT",CP_FIXED,"VIN",CP_FIXED,"VOUT",CP_FIXED,
                        "IBIAS",CP_FIXED,"VP1",CP_NO) ;

CAIRO_DECLARE_PROCEDURE("UGF",CP_COMPLETE,"TEMP",CP_FIXED,"CLOAD",CP_FIXED,"UGF",CP_INF,"PM",CP_RANGE,
                        "IBIAS",CP_NO,"SR",CP_INF,
                        "GAIN",CP_INF,"COF",CP_NO,"GBW",CP_INF) ;
```

Les messages associés aux codes de retour de ces procédures sont à leur tour déclarés :

```
  // Messages de VP1
  CAIRO_DECLARE_PROCEDURE_MESSAGE(1 ,"VP1: DP: IBIAS is too large, try to decrease current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(2 ,"VP1: DP: IBIAS is too small, try to increase current") ;
  ...

  // Messages de UGF
  CAIRO_DECLARE_PROCEDURE_MESSAGE(10,"UGF: IBIAS is too large");
  CAIRO_DECLARE_PROCEDURE_MESSAGE(11,"UGF: IBIAS is too small");
  ...
CAIRO_END_CREATE(OTA)
```

Procédures de dimensionnement UGF

La procédure $VP1$ a illustré les calculs DC (voir le chapitre précédent). On présente ici la procédure UGF qui fait intervenir les paramètres petits signaux. Le code a été volontairement simplifié.

La procédure UGF utilise la procédure $VP1$ qui a calculé les paramètres petits signaux (la procédure $VP1$ doit donc avoir été appelée avant UGF). Ceux-ci, mémorisés dans des variables locales, sont rappelés :

```
CAIRO_BEGIN_DSES(OTA,char *name)

  double gds_bias     = 0.0 ;
  double cgd_dp       = 0.0 ;
  double cbd_dp       = 0.0 ;
  ...

  CAIRO_BEGIN_PROCEDURE("UGF")

    CAIRO_COMPUTE("VP1") ;

    CAIRO_GET_LOCAL_VARIABLE("gds_bias",gds_bias) ;
    CAIRO_GET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
    CAIRO_GET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;
   ...

  END_PROCEDURE

CAIRO_END_DSES(OTA)
```

La boucle principale de calcul de UGF consiste à utiliser ces paramètres petits signaux pour déterminer la pulsation au point de transition, laquelle est à son tour utilisée pour calculer UGF :

```
  CAIRO_BEGIN_PROCEDURE("UGF")

    // Initialisation: on se base sur les données du point de polarisation

    ids_bias = sign * ibias_cal ;
    ids_dp = sign * ibias_cal ;
    ids_cm = - sign * (ibias_cal / 2.0) ;

    // Calcul des parametres intermédiaires
    fw0w1w2G1G2C2(gds_dp,cgd_dp,cbd_dp,
                  gds_cm,gm_cm,cgs_cm,cgb_cm,cbd_cm,cgd_cm,
                  cload_cal,
                  w0,w1,w2,G1,G2,C2) ;

    // Calcul de la pulsation au point de transition
    wt = fwt(gm_dp,gm_cm,G1,G2,w0,w1,w2,C2) ;
    ugf_cal = fugf(wt) ;

    // Produit gain-bande
    gbw_cal = fgbw(gain_cal_lin,w2) ;

    ...
  END_PROCEDURE
```

Notons que *UGF* fait appel à des fonctions utilitaires que sont fw0w1w2G1G2C2, fwt et fgbw. Par exemple, fw0w1w2G1G2C2 utilise un modèle approximé mais dont la précision est suffisante :

```
static void fw0w1w2G1G2C2(double gds_dp,double cgd_dp,double cbd_dp,
                          double gds_cm,double gm_cm,double cgs_cm,double cgb_cm,double cbd_cm,
                          double cgd_cm,double cload,
                          double &w0,double &w1,double &w2,double &G1,double &G2,double &C2)
{
  double C1 = 0.0 ;

  // Elements intermédiaires
  C1 = cgd_dp + cbd_dp + cgs_cm + cgb_cm + cbd_cm + cgs_cm + cgb_cm ;
  C2 = cgd_dp + cbd_dp + cgd_cm + cbd_cm + cload ;

  G1 = gds_dp + gds_cm + gm_cm ;
  G2 = gds_dp + gds_cm ;
```

```
  // Calcul des pôles
  w1 = G1 / C1 ;

  w2 = (gds_dp + gds_cm) / C2 ;

  // Zero
  w0 = w1 * (1.0 + (gm_cm / G1)) ;
}
```

Enfin, la boucle principale permettant de satisfaire la fréquence unitaire demandée augmente le courant de polarisation si nécessaire, puis rappelle la procédure $VP1$:

```
    do
      {
        delta_ugf = fabs((ugf_in - ugf_cal) / ugf_cal) ;

        // Inutile d'avoir une precision trop grande, ce qui peut poser des problèmes
        // de convergence
        ugf_cal_mhz = ugf_cal / 1.0e6 ;

        if((ugf_cal_mhz < ugf_in_mhz) || (delta_ugf > epsilon2))
          {
            // Aucune contrainte n'est satisfaite (ni ugf >= ugf_in, ni ugf = ugf_in, moyennant epsilon2)

            loop_ugf = true ;
            l_bias = l_min ;
            l_dp = l_min ;
            l_cm = l_min ;

            CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
            CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
            CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

            ibias_cal = ibias_cal * (ugf_in / ugf_cal) ; // ugf_in / ugf_cal > 1

            CAIRO_SET_PARAM("IBIAS",ibias_cal) ;

            CAIRO_COMPUTE("VP1") ;
          }
        else
          {
            // on a ugf >= ugf_in, on arrête
            loop_ugf = false ;
            ugf_satisfied = true ;
          }

      }while(ugf_satisfied == false) ;
```

On remarquera l'enchaînement classique : positionnement des paramètres (CAIRO_SET_PARAM) et appel à la procédure (CAIRO_COMPUTE).

9.1.7 Simulations et résultats

Dans ce paragraphe nous présentons les résultats de dimensionnement obtenus avec la procédure UGF. La plateforme de tests peut-être consultée en annexe A. La validation des résultats s'effectue en deux étapes :

1. D'abord on valide le dimensionnement par simulation électrique, sans prendre en compte les phénomènes liés au dessin des masques (capacités parasites, etc.). Le

simulateur utilise le schéma électrique dimensionné généré par CAIRO+ à partir des spécifications données ; les performances obtenues sont comparées avec, d'une part, les spécifications imposées à l'entrée du générateur et, d'autre part, les performances estimées par la procédure de dimensionnement.

Plus que la procédure de dimensionnement à proprement parler, ce sont les méthodes d'estimation des performances qui sont validées par cette étape. En effet, dans une approche basée sur la connaissance, les performances sont toujours estimées à l'aide d'équations approchées, ce qui représente une source potentielle d'imprécision. Alors, même si les spécifications de départ semblent atteintes, une vérification est nécessaire afin de s'assurer que le circuit dimensionné réel les respecte effectivement.

2. Après avoir dimensionné et vérifié un circuit électrique, il faut s'assurer qu'une fois implanté sur silicium, ce circuit respecte toujours les spécifications de départ. Pour cela, à partir du dessin des masques fourni par CAIRO+, on réalise une extraction du circuit électrique *à l'aide un outil d'extraction indépendant*, et l'on soumet le schéma électrique extrait au simulateur. On compare ensuite les spécifications du circuit extrait avec les spécifications de départ et avec les performances estimées à l'étape du dimensionnement.

On illustre la puissance du générateur en l'utilisant dans deux contextes : dans le premier cas, deux circuits avec les mêmes spécifications doivent être réalisés dans deux technologies différentes (« migration technologique »), dans l'autre cas, deux circuits doivent être réalisés dans la même technologie mais avec les spécifications différentes (« migration des spécifications »).

Migration technologique

L'OTA a été généré avec les mêmes spécifications électriques d'entrée pour deux technologies différentes, $0,6\ \mu m$ et $0,13\ \mu m$.

Le tableau 9.2 présente les valeurs des paramètres d'entrée et de sortie de l'OTA ainsi que les valeurs des dimensions des transistors après le dimensionnement électrique.

Le tableau 9.3 compare les valeurs attendues de la caractérisation des performances avec les valeurs obtenues par simulations. Sont simulés :

- Le schéma électrique dimensionné généré par CAIRO+. Les calculs ont été effectués sans tenir compte des capacités parasites de routage. En revanche, les capacités parasites internes des transistors ont été prises en compte dès le dimensionnement électrique, ce qui influe directement sur les W et L de ces transistors.
- Le schéma électrique dimensionné issu de l'extraction du dessin des masques généré par CAIRO+ où les capacités parasites de routage ne sont pas extraites.
- Le schéma électrique dimensionné issu de l'extraction du dessin des masques généré par CAIRO+ où les capacités parasites de routage sont extraites (les capacités de routage sont des capacités par rapport au substrat).

L'écart relatif donné en pourcentage entre parenthèses est mesuré par rapport aux calculs de CAIRO+.

TAB. 9.2: Valeur des paramètres de l'OTA

Paramètre d'entrée	Technologie 0,6 μm	Technologie 0,13 μm
UGF (MHz)	55,0	55,0
PM (degré)	73 - 76	73 - 76
TEMP (kelvin)	300,15	300,15
V_{IN} (V)	1,65	0,6
V_{OUT} (V)	1,65	0,6
$V_{EG_{BIAS}}$ (V)	0,3	0,1
$V_{EG_{DP}}$ (V)	0,3	0,1
C_{LOAD} (pF)	5,0	5,0
V_{DD} (V)	3,3	1,2
V_{SS} (V)	0	0

Paramètre de sortie		Technologie 0,6 μm	Technologie 0,13 μm
UGF (MHz)		57,25	55,38
PM (degré)		75,88	75,39
GAIN (db)		42,36	45,28
GBW (MHz)		64,53	62,85
COF (MHz)		0,49	0,34
SR (V/μs)		137,10	54,66
V_{P1} (V)		2,141	0,759835
I_{BIAS} (μA)		877,545	292,761
BIAS	L (μm)	2,9	2,615
	W (μm)	1756,43	1612,46
DP	L (μm)	2,9	2,615
	W (μm)	856.431	786,589
CM	L (μm)	2,9	2,615
	W (μm)	47,289	27,8837

TAB. 9.3: Performances comparées de l'OTA : UGF (MHz), PM (degré) et GAIN (db)

Tech.	Paramètre	Spéc.	Calcul	Simulation du schéma électrique dimensionné (écart relatif/calcul %)		
				Généré par CAIRO+	Extrait du *layout*	
					Sans C rout.	Avec C rout.
0.6 μm	UGF	55,0	57,25	59,00 (+3,06)	63,70 (+11,27)	61,90 (+8,12)
	PM	73 - 76	75,88	71,62 (-5,61)	73,34 (-3,35)	72,77 (-4,10)
	GAIN	-	42,36	42,40 (+0,09)	42,40 (+0,09)	42,40 (+0,09)
0.13 μm	UGF	55,0	55,38	55,10 (-0,51)	55.10 (-0,51)	54,10 (-2,31)
	PM	73 - 76	75,39	71,62 (-5,00)	71,62 (-5,00)	71,05 (-5,76)
	GAIN	-	45,28	45,30 (+0,04)	45,30 (+0,04)	45,30 (+0,04)

- Les calculs menés par CAIRO+ (colonne « Calcul » du tableau 9.3) satisfont les contraintes imposées. La fréquence de transition a été réglée pour être comprise dans un intervalle de ± 15 % de la valeur demandée. La marge de phase est proche de la borne haute de l'intervalle car celle-ci, toujours élevée à l'initialisation, l'atteint par décroissance.
- Le gain statique obtenu par mesure à l'issue de la simulation des schémas électriques dimensionnés (colonnes « Schéma électrique généré par CAIRO+ » et « Schéma électrique extrait du *layout* » est toujours très précis par rapport au gain statique obtenu par calcul, ce qui valide les équations en régime statique.
- L'erreur sur la marge de phase obtenue par simulation reste contenue quelle que soit la technologie (erreur relative de 5 % au maximum). La valeur obtenue par simulation est toujours inférieure à la valeur prévue par le calcul, probablement en raison d'effets non pris en compte dans le modèle des transistors qui a été implanté dans CAIRO+.
- Enfin, les valeurs obtenues par simulation pour la fréquence de transition montrent que celle-ci est supérieure à la spécification de départ en 0,6 μm et légèrement inférieure en 0,13 μm.

Le modèle électrique implanté, ainsi que la qualité des données du fichier technologique, influent sur les résultats. Pour s'en convaincre, le tableau 9.4 compare les paramètres petits signaux obtenus par CAIRO+ pour tous les dispositifs élémentaires (uniquement ceux intervenant dans les équations) avec les paramètres petits signaux donnés par le simulateur électrique d'après le schéma électrique dimensionné généré par CAIRO+. L'écart très faible en $0,6$ μm et en $0,13$ μm indique que le modèle électrique implanté dans CAIRO+ est fiable.

Les figures 9.9 présentent les simulations réalisées à partir du schéma électrique dimensionné extrait avec les capacités parasites de routage en 0.13 μm. Les figures de gauche donnent l'évolution du gain en fonction de la fréquence (la fréquence de transition est celle pour lequel le gain en db est nul), celles de droite donnent la phase en fonction de la fréquence.

L'utilisation de dispositifs élémentaires travaillant directement avec des dimensions réelles (et non au lambda) permet de conserver une excellente précision dans les dimensions effectivement générées des transistors quelle que soit la technologie utilisée (tableau 9.5). Le nombre de repliements, pour lequel une erreur cumulative sur le W s'ajoute en proportion, est contrebalancé par le pas de grille réduit dans les technologies submicroniques.

La figure 9.10 présente les deux dessins des masques des circuits générés. Les proportions et les tailles relatives ont été respectées de manière à pouvoir comparer l'évolution de la finesse de gravure. Le facteur de forme choisi est le carré.

TAB. 9.4: Comparaison des paramètres petits signaux de l'OTA

Technologie	Paramètre		Calcul	Simulation	Écart (%)
0.6 μm	BIAS	i_{DS}	-0,000877545	-8,77905263E-04	+0,04
		g_{DS}	5,92347e-05	5,95710718E-05	+0,57
	DP	i_{DS}	-0,000438773	-4,38948604E-04	+0,04
		v_{TH}	-0,858342	-8,58343771E-01	+0,0
		c_{GD}	2,90739e-13	2,90583183E-13	-0,05
		c_{BD}	1,03521e-12	1,03345115E-12	-0,17
		gm	0,00262066	2,62099434E-03	+0,01
		gmb	0,000759483	7,59580005E-04	+0,01
		g_{DS}	1,23257e-05	1,24130390E-05	+0,71
	CM	i_{DS}	0,000438773	4,38956494E-04	+0,04
		c_{GS}	1,72136e-13	1,71823845E-13	-0,18
		c_{GB}	3,37965e-14	3,37254807E-14	-0,21
		c_{BD}	5,83793e-14	5,81614527E-14	-0,37
		c_{GD}	1,60688e-14	1,60381694E-14	-0,19
		gm	0,00100131	1,00042230E-03	-0,09
		g_{DS}	7,44728e-06	7,43970476E-06	-0,10
0.13 μm	BIAS	i_{DS}	-0,000292761	-2,9275E-04	-0,004
		g_{DS}	0,000117612	1,1759E-04	-1,456
	DP	i_{DS}	-0,000146381	-1,4637E-04	-0,008
		v_{TH}	-0,340186	-3,4019E-01	+0,001
		c_{GD}	3,33928e-13	3,3388E-13	-0,014
		c_{BD}	7,28323e-15	7,2722E-15	-0,151
		gm	0,00212901	2,1289E-03	-0,005
		gmb	0,000493237	4,9322E-04	-0,003
		g_{DS}	4,45682e-06	4,4546E-06	-0,050
	CM	i_{DS}	0,000146381	1,4637E-04	-0,008
		c_{GS}	6,34396e-13	6,3518E-13	+0,124
		c_{GB}	3,75666e-14	3,7628E-14	+0,163
		c_{BD}	5,07236e-16	5,0853E-16	+0,255
		c_{GD}	1,46695e-14	1,4689E-14	+0,133
		gm	0,000885881	8,8641E-04	+0,060
		g_{DS}	7,05758e-06	7,0623E-06	+0,067

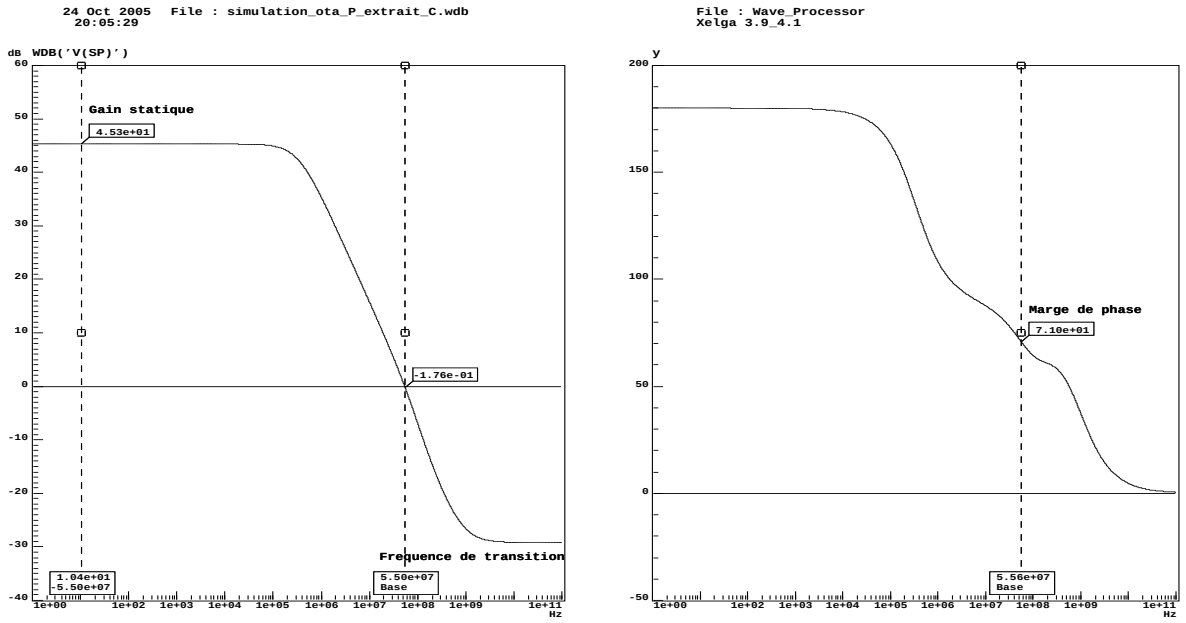


FIG. 9.9: Évolutions du gain (à gauche) et de la phase (à droite) en fonction de la fréquence en 0.13 μm

TAB. 9.5: Écart dans les tailles calculées et effectivement générées des transistors

Technologie	Paramètre			Calcul	Génération	Écart relatif (%)
0.6 μm	BIAS	$M = 28$	W (μm)	1756,43	1757,0	+0,03
			L (μm)	2,9	2,9	+0,0
	DP	$M = 12$	W (μm)	856,431	856,20	+0,03
			L (μm)	2,9	2,9	+0,0
	CM	$M = 8$	W (μm)	47,289	47,20	-0,19
			L (μm)	2,9	2,9	+0,0
0.13 μm	BIAS	$M = 33$	W (μm)	1612,46	1612.38	-0,005
			L (μm)	2,615	2,615	+0,0
	DP	$M = 16$	W (μm)	786,589	786,56	-0,004
			L (μm)	2,615	2,615	+0,0
	CM	$M = 16$	W (μm)	27,8837	27,92	+0,130
			L (μm)	2,615	2,615	+0,0

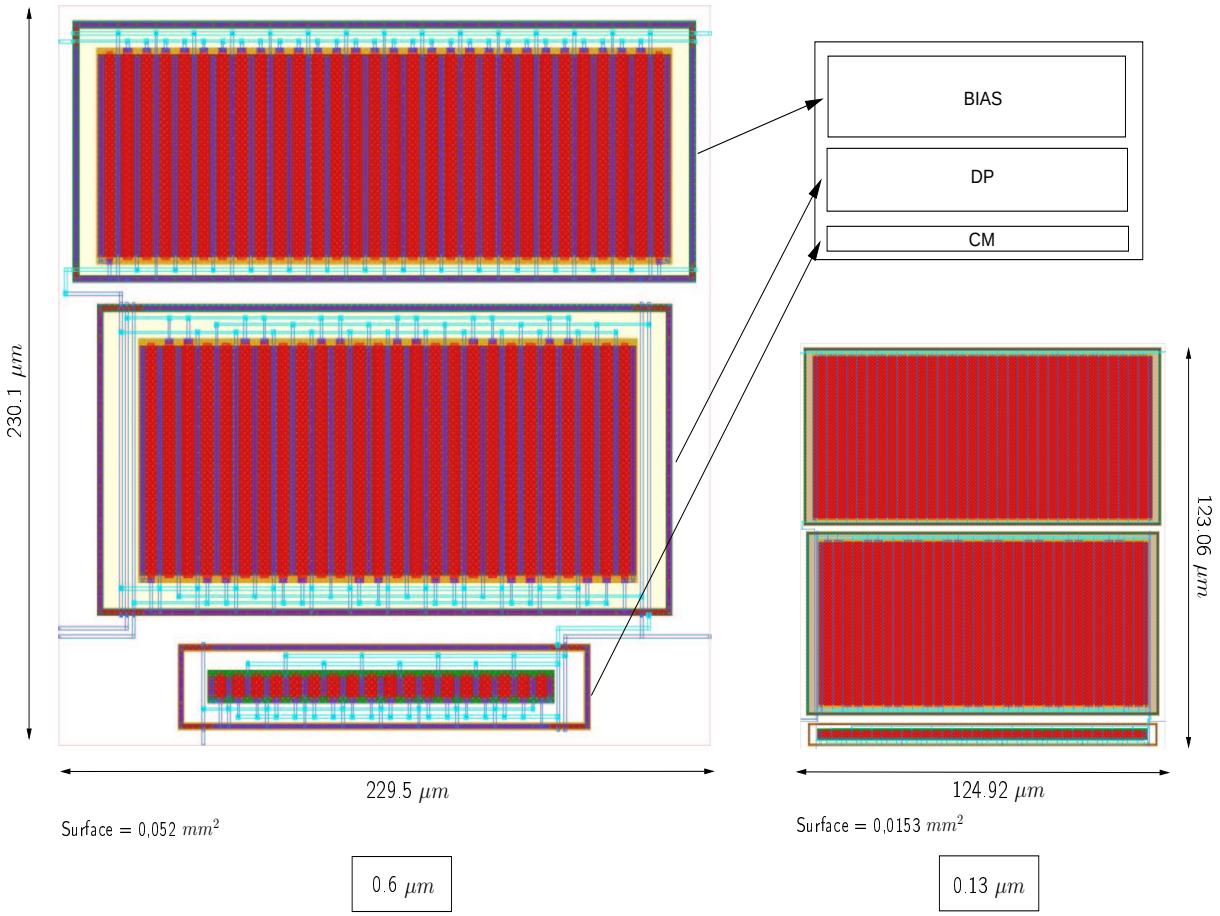


FIG. 9.10: Dessin des masques des deux OTA en $0,6\ \mu m$ et $0,13\ \mu m$ (échelle relative respectée)

Migration de spécification

Reprenant le même OTA, celui-ci a été généré avec deux spécifications électriques d'entrée différentes se situant dans la borne haute des possibilités offertes par la technologie $0,13\ \mu m$.

Le tableau 9.6 présente les valeurs des paramètres d'entrée et de sortie de l'OTA, ainsi que les valeurs des dimensions des transistors après le dimensionnement électrique. Les spécifications sont en général satisfaites. L'écart relatif constaté entre les mesures et les calculs est faible (tableau 9.7). L'effet des capacités parasites de routage est sensible, faisant diminuer de quelques MHz la fréquence de transition réellement obtenue.

Les dimensions effectivement générées des transistors ont été respectées (tableau 9.8). La figure 9.11 présente les diagrammes de Bode du gain et de la phase pour les deux spécifications en $0,13\ \mu m$.

TAB. 9.6: Valeur des paramètres de l'OTA

Paramètre d'entrée	Spécification 1	Spécification 2
UGF (MHz)	200,0	300,0
PM (degré)	73 - 76	73 - 76
TEMP (kelvin)	300,15	300,15
V_{IN} (V)	0,6	0,6
V_{OUT} (V)	0,6	0,6
$V_{EG_{BIAS}}$ (V)	0,1	0,1
$V_{EG_{DP}}$ (V)	0,1	0,1
C_{LOAD} (pF)	5,0	5,0
V_{DD} (V)	1,2	1,2
V_{SS} (V)	0	0

Paramètre de sortie		Spécification 1	Spécification 2
UGF (MHz)		202,68	304,84
PM (degré)		75,99	75,98
GBW (MHz)		228,19	344,06
COF (MHz)		1,93	3,37
SR ($V/\mu s$)		200,19	303,22
GAIN (db)		41,47	40,17
V_{P1} (V)		0.756361	0.754386
I_{BIAS} (mA)		1,11794	1,72394
BIAS	L (μm)	1,21	0,945
	W (μm)	2749,66	3251,41
DP	L (μm)	1,21	0,945
	W (μm)	1324,09	1555,05
CM	L (μm)	1,21	0,945
	W (μm)	56,9838	73,7278

TAB. 9.7: Performances comparées de l'OTA : UGF (MHz), PM (degré) et GAIN (db)

Tech.	Paramètre	Spéc.	Calcul	Simulation du schéma électrique dimensionné (écart relatif/calcul %)		
				Généré par CAIRO+	Extrait du *layout*	
					Sans C rout.	Avec C rout.
Spec 1	UGF	200,0	202,68	201,5 (-0,58)	201,5 (-0,58)	199,0 (-1,81)
	PM	73 - 76	75,99	70,47 (-7,26)	70,47 (-7,26)	70,47 (-7,26)
	GAIN	-	41,63	41,50 (-0,31)	41,50 (-0,31)	41,50 (-0,31)
Spec 2	UGF	300,0	304,84	303,0 (-0,60)	304,0 (-0,28)	297,5 (-2,41)
	PM	73 - 76	75,98	69,90 (-8,00)	69,90 (-8,00)	69,33 (-8,76)
	GAIN	-	40,17	40,20 (-0,07)	40,20 (-0,07)	40,20 (-0,07)

TAB. 9.8: Écart dans les tailles calculées et effectivement générées des transistors

Spéc	Paramètre			Calcul	Génération	Écart relatif (%)
Spéc 1	BIAS	$M = 56$	W (μm)	2749,66	2749,88	+0,008
			L (μm)	1,21	1,21	+0,0
	DP	$M = 26$	W (μm)	1324,09	1324,05	-0,003
			L (μm)	1,21	1,21	+0,0
	CM	$M = 16$	W (μm)	56,9838	56,96	-0,042
			L (μm)	1,21	1,21	+0,0
Spéc 2	BIAS	$M = 64$	W (μm)	3251,41	3251,52	+0,003
			L (μm)	0,945	0,945	+0,0
	DP	$M = 32$	W (μm)	1555,05	1555,04	-0,001
			L (μm)	0,945	0,945	+0,0
	CM	$M = 32$	W (μm)	73,7278	73,76	+0,044
			L (μm)	0,945	0,945	+0,0

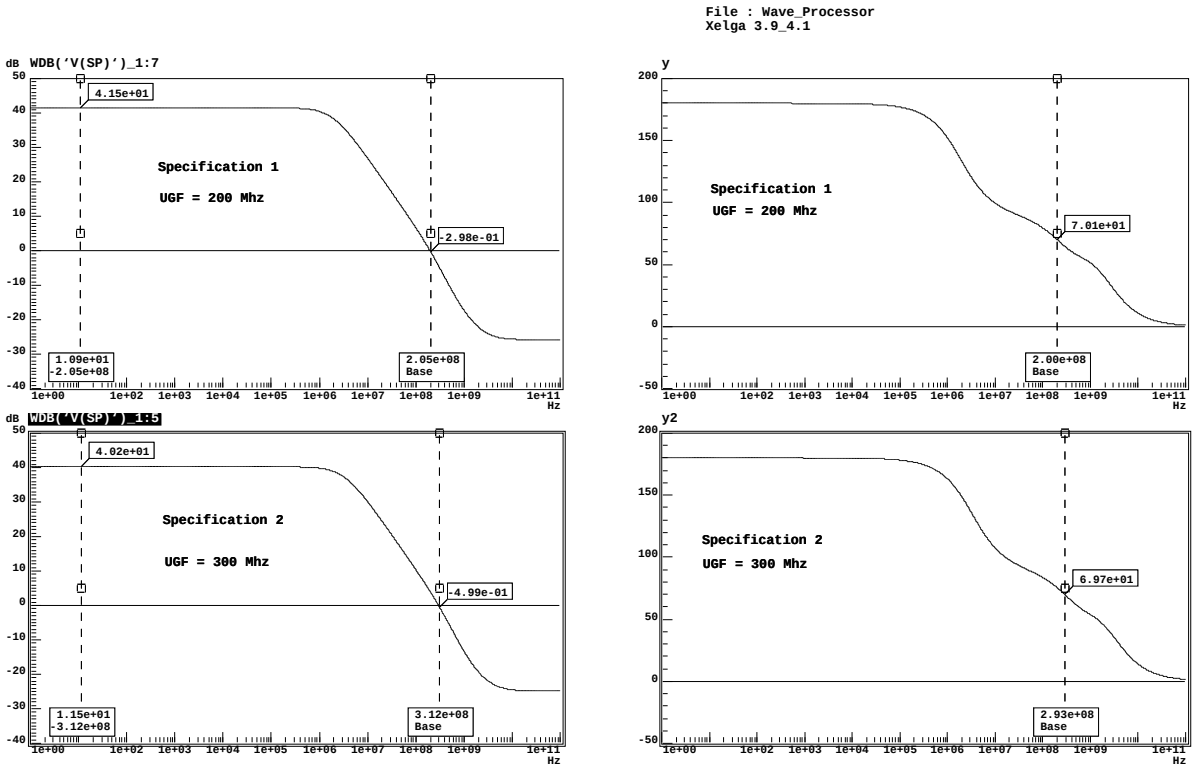


FIG. 9.11: Fréquence de transition et phase pour le schéma électrique dimensionné extrait avec capacités parasites de routage pour les deux spécifications en $0,13\ \mu m$

Facteurs de forme et modes de réalisation

La figure 9.12 présente l'OTA réalisé pour la spécification 2 (une fréquence de transition de 300 MHz) en technologie $0,13\ \mu m$ avec à gauche un facteur de forme différent du carré,

à droite un nombre d'empilements fixé à trois pour la paire différentielle. Les autres modes de réalisation (transistors fictifs et anneaux de garde) sont identiques.

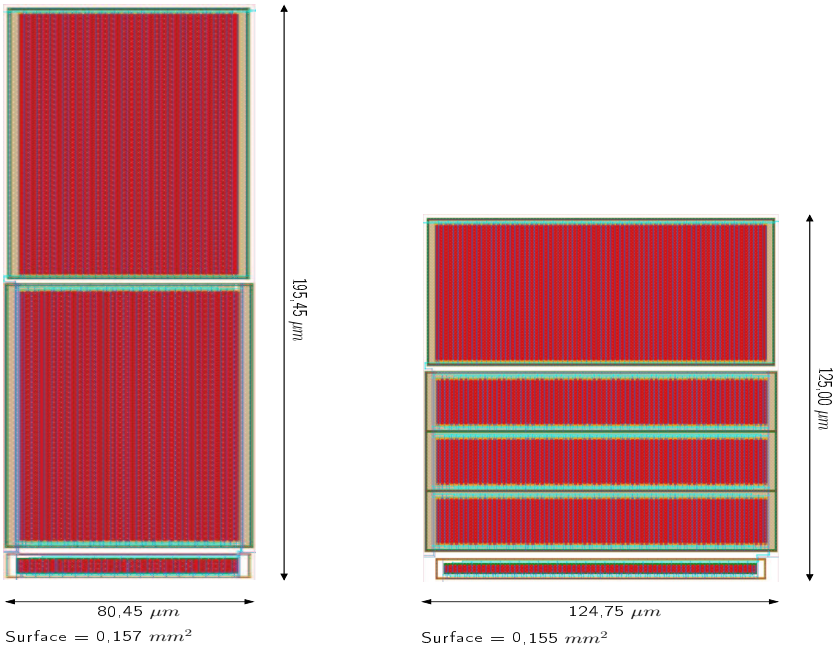


FIG. 9.12: OTA pour deux facteurs de forme et deux modes de réalisation

9.2 Conception hiérarchique d'un filtre passe-bas pour un modulateur ΣΔ à temps continu et mode courant

Ce paragraphe rappelle brièvement l'architecture et le fonctionnement d'un modulateur ΣΔ à temps continu. Le lecteur désireux d'approfondir le sujet pourra consulter la thèse d'Hassan Aboushady [Abo02] qui a servi de support à la réalisation de l'intégrateur de courant.

9.2.1 Objectif et architecture

Un modulateur ΣΔ a pour fonction de mettre en forme le bruit accompagnant la quantification d'un signal de manière à le rejeter en dehors de la bande de fréquence utile. La figure 9.13 montre la répartition typique de la densité spectrale du signal en fonction de la fréquence à la sortie d'un modulateur. On peut remarquer que le bruit a été amplifié dans une bande de fréquence ne contenant pas le signal utile, et atténué dans la bande utile.

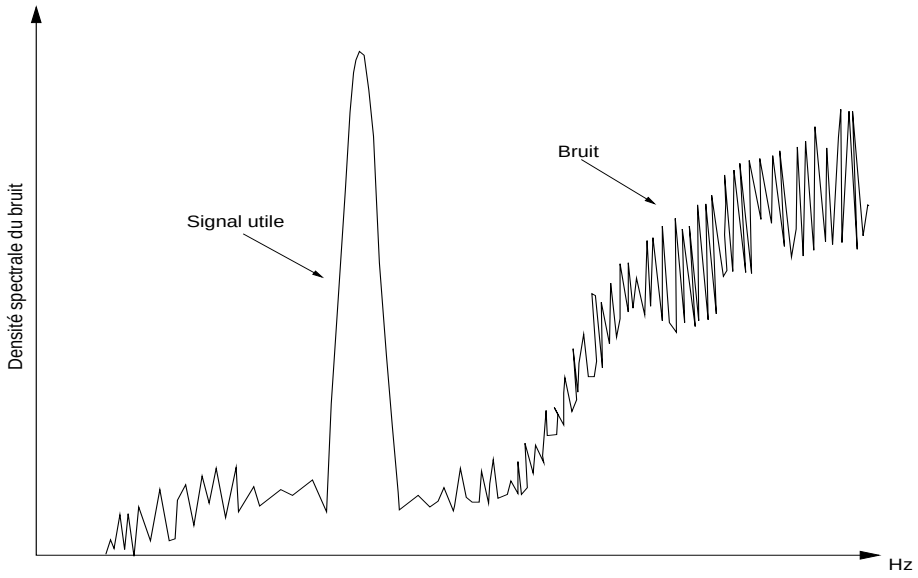


FIG. 9.13: Densité spectrale typique d'un modulateur ΣΔ

Le circuit étudié ici repose sur un comparateur et un filtre passe-bas réalisé par trois intégrateurs de courant en cascade, intégrateurs dont les cœfficients sont calculés sous *matlab* lors de la phase de conception système [Abo02]. CAIRO+ intervient dans le dimensionnement des intégrateurs (figure 9.14). Le filtre passe-bas constitué par la mise en cascade des trois intégrateurs, forme un module de niveau 2.

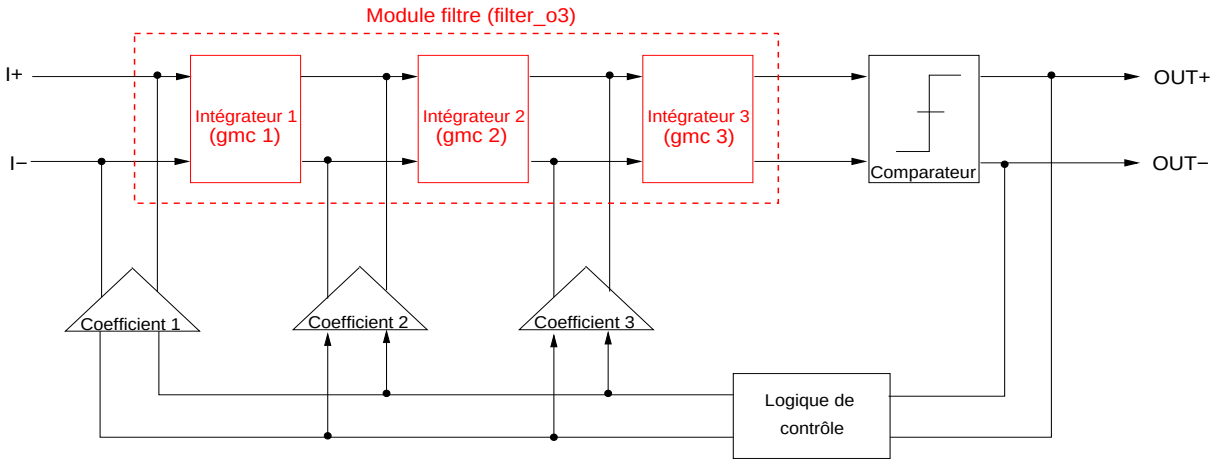


FIG. 9.14: Architecture d'un modulateur ΣΔ à temps continu d'ordre 3

Chaque intégrateur de courant est un intégrateur *gmc* d'ordre 1 fonctionnant en mode courant. Ils sont théoriquement basés sur des miroirs de courant simples (figure 9.15). Il a été décidé ici de n'utiliser que des transistors simples de manière à favoriser un placement symétrique.

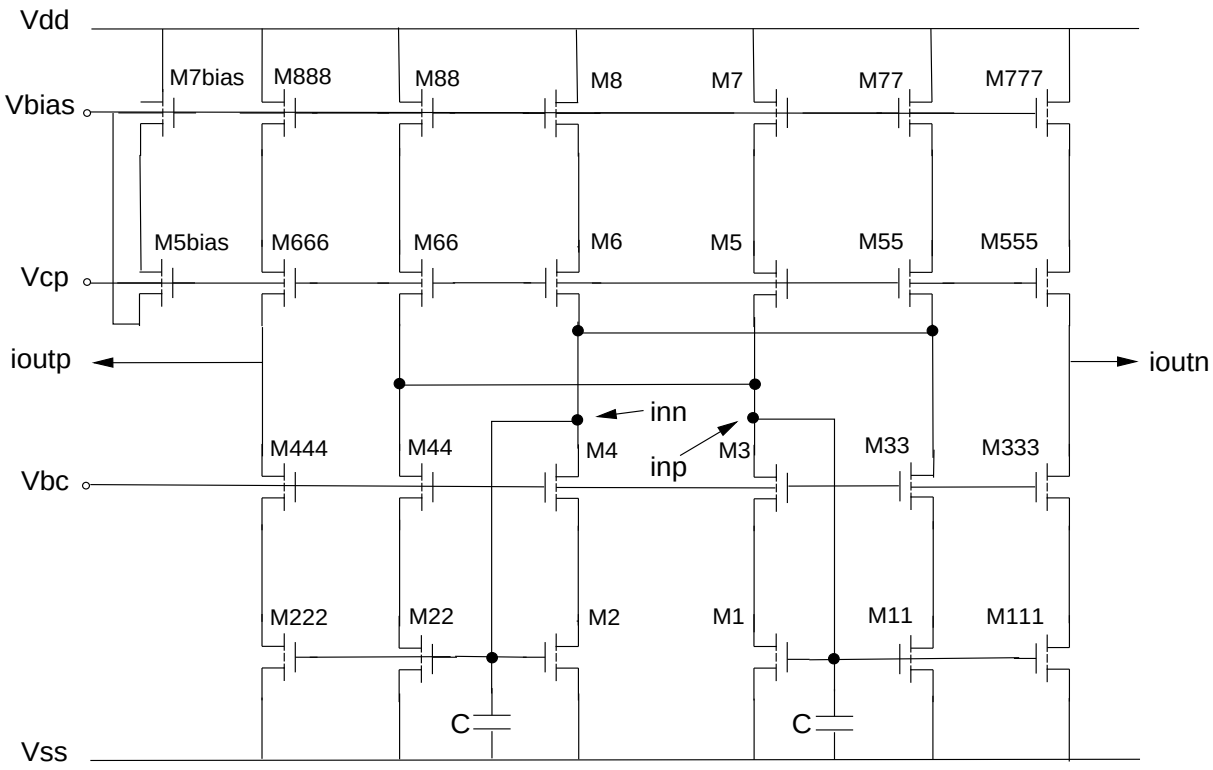


FIG. 9.15: Topologie d'un intégrateur de courant d'ordre 1

9.2.2 Hiérarchie d'un module « Filtre passe-bas »

Le module filtre est formé par la mise en cascade de trois intégrateurs nommés *gmc* (des modules de niveau 1). Chaque intégrateur est constitué d'un amplificateur et d'une paire de capacités. Chaque amplificateur *gmc* contient vingt-quatre transistors simples (figure 9.16), plus les deux transistors chargés de la polarisation. Un seul générateur de *gmc* est donc nécessaire : il est instancié trois fois avec des spécifications électriques différentes.

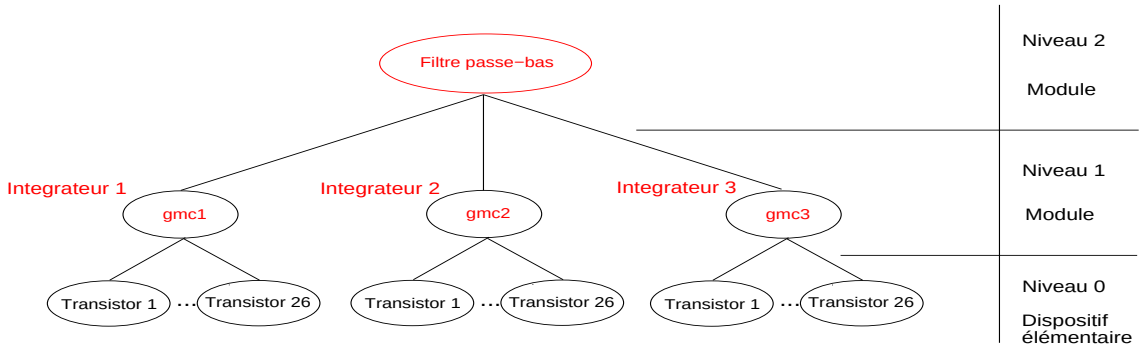


FIG. 9.16: Hiérarchie d'un générateur d'intégrateur d'ordre 1

9.2.3 Dimensionnement électrique

Paramètres du module de filtre de niveau 2

Les paramètres du tableau 9.9 sont transmis tels quels aux trois intégrateurs. Les rapports des courant (RATIO_X) ne sont pas pris en compte si la consommation est contrainte (voir paragraphe suivant).

TAB. 9.9: Paramètres du filtre passe-bas

	Paramètre	Type de la contrainte	Signification
Entrées	TEMP	CP_FIXED	Température
	RESOLUTION	CP_FIXED	Résolution en nombre de bits équivalents
	BW	CP_FIXED	Largeur de bande
	RATIO_1	CP_FIXED	Rapport des courants
	RATIO_2	CP_FIXED	Rapport des courants
	RATIO_3	CP_FIXED	Rapport des courants
	AINT1_1	CP_FIXED	Cœfficient du premier intégrateur
	AINT1_2	CP_FIXED	Cœfficient du deuxième intégrateur
	AINT1_3	CP_FIXED	Cœfficient du troisième intégrateur
	OSR	CP_FIXED	Rapport de sur-échantillonnage
	ASD	CP_FIXED	Amplitude du signal
	VEGM3	CP_FIXED	V_{eg} du transistor M3
	VEGM5	CP_FIXED	V_{eg} du transistor M5
	LM1	CP_FIXED	Longueur du transistor M1
	LM3	CP_FIXED	Longueur du transistor M3
	LM5	CP_FIXED	Longueur du transistor M5
	LM7	CP_FIXED	Longueur du transistor M7
Sorties	POWER_COMSUMPTION	CP_SUP	Consommation (optionnel)
	VBC	CP_NO	Tension de polarisation des miroirs de courant
	VCP	CP_NO	Tension de polarisation
	VBIAS	CP_NO	Tension de polarisation des intégrateurs
	CAPINT_1	CP_NO	Capacité d'intégration de l'intégrateur 1
	IBIAS_1	CP_NO	Courant de polarisation de l'intégrateur 1

Fin du tableau page suivante...

	Paramètre	Type de la contrainte	Signification
Sorties	MODULATION_COEFF_1	CP_NO	Indice de modulation de l'intégrateur 1
	CAPINT_2	CP_NO	Capacité d'intégration de l'intégrateur 2
	IBIAS_2	CP_NO	Courant de polarisation de l'intégrateur 2
	MODULATION_COEFF_2	CP_NO	Indice de modulation de l'intégrateur 2
	CAPINT_3	CP_NO	Capacité d'intégration de l'intégrateur 3
	IBIAS_3	CP_NO	Courant de polarisation de l'intégrateur 3
	MODULATION_COEFF_3	CP_NO	Indice de modulation de l'intégrateur 3

Contrainte possible applicable au filtre

La consommation peut être contrainte (tableau 9.10). Dans ce cas, si la consommation trouvée en utilisant les rapports initiaux fournis par l'utilisateur ne satisfait pas la contrainte, de nouveaux rapports sont calculés par la procédure de dimensionnement du filtre.

TAB. 9.10: Contraintes sur les paramètres

Paramètre	Contraintes
POWER_CONSUMPTION	POWER $\leqslant$ POWER_max (facultatif)

Paramètres du mode intégrateur de niveau 1

Les paramètres du générateur d'intégrateur sont résumés dans le tableau 9.11.

Procédure de dimensionnement du module filtre

Le filtre passe-bas ne possède qu'une seule procédure complète de dimensionnement, SIZE_FILTER_O3, tout comme les intégrateurs *gmc* dont le nom de la procédure est SIZE_GMC.

Le déroulement du dimensionnement du filtre tient compte de la contrainte facultative sur la consommation. Si celle-ci est imposée et que la consommation obtenue en prenant

TAB. 9.11: Paramètres de l'intégrateur

	Paramètre	Type de la contrainte	Signification
Entrées	TEMP	CP_FIXED	Température
	RATIO	CP_FIXED	Rapport des courants
	BW	CP_FIXED	Largeur de bande
	AINT	CP_FIXED	Cœfficient de l'intégrateur
	OSR	CP_FIXED	Rapport de sur-échantillonnage
	ASD	CP_FIXED	Amplitude du signal
	VEGM3	CP_FIXED	V_{eg} du transistor M3
	VEGM5	CP_FIXED	V_{eg} du transistor M5
	LM1	CP_FIXED	Longueur du transistor M1
	LM3	CP_FIXED	Longueur du transistor M3
	LM5	CP_FIXED	Longueur du transistor M5
	LM7	CP_FIXED	Longueur du transistor M7
Sorties	UGF	CP_NO	Fréquence de coupure
	SIGNAL_POWER	CP_NO	Puissance du signal
	CAPINT	CP_NO	Capacité d'intégration de l'intégrateur
	IBIAS	CP_NO	Courant de polarisation de l'intégrateur
	VBC	CP_NO	Tension de polarisation du miroir de courant
	VCP	CP_NO	Tension de polarisation
	VBIAS	CP_NO	Tension de polarisation de l'intégrateur
	MODULATION_COEFF	CP_NO	Indice de modulation

les rapports de courant initiaux donnés par l'utilisateur ne satisfait pas la contrainte, le rapport du courant du *gmc1* est positionné à 1 tandis que l'algorithme essaie toutes les combinaisons de rapport des *gmc2* et *gmc3* en veillant à ce que RAPPORT_2 > RAPPORT_3 et en faisant décroître les rapports par pas de 0,1 à partir de 1 (on a donc les rapports (1 ;1 ;1) (1 ;1 ;0.9) (1 ;1 ;0.8) ... (1 ;0.9 ;0.9) (1 ;0.9 ;0.8) ... (1 ;0.1 ;0.1)). Les étapes du dimensionnement sont les suivantes :

1. Récupérer les paramètres du filtre
2. Pour chaque intégrateur *gmc*
   (a) Positionner les paramètres d'entrée du *gmc*
   (b) Appeler la procédure SIZE_GMC
   (c) Récupérer les paramètres de sortie du *gmc*
3. Calculer les paramètres de sortie du filtre (dont POWER_cal)

4. Si une consommation maximale POWER_max est imposée
   (a) Si POWER_cal > POWER_max
      i. Si ratio_2 > ratio_min
         A. Si ratio_3 ≤ ratio_min
            - ratio_2 = ratio_2 - ratio_step
            - ratio_3 = ratio_2
            - Aller en 2
         B. Sinon ratio_3 = ratio_3 - ratio_step
            - Aller en 2
      ii. Sinon la consommation ne peut être satisfaite
5. Rendre les paramètres de sortie du filtre

Procédure de dimensionnement du module intégrateur

Chaque intégrateur *gmc* utilise une procédure de dimensionnement électrique basée sur [Abo02] :

1. Récupérer les paramètres d'entrée
2. Calculer l'indice de modulation
3. Calculer la capacité d'intégration
4. Calculer le courant IDS
5. Positionner les paramètres de la procédure "W(VGS,L,IDS)" pour M1
6. Répéter tant que PROCEDURE_STATUS ≠ CP_OK
   (a) Positionner le paramètre L de "W(VGS,L,IDS)"
   (b) Calculer les W des transistors avec "W(VGS,L,IDS)"
   (c) Examiner le code de retour de la procédure
      i. Si W est trop grand, arrêter la boucle
      ii. Si W est trop petit, augmenter la largeur L du transistor
   (d) Récupérer W
7. Positionner les paramètres de la procédure "W(VEG,L,IDS)" pour M3
8. Répéter tant qu'il y a un échec
   (a) Positionner le paramètre L de "W(VEG,L,IDS)"
   (b) Calculer les W des transistors avec "W(VEG,L,IDS)"
   (c) Examiner le code de retour de la procédure
      i. Si W est trop grand, arrêter la boucle
      ii. Si W est trop petit, augmenter la largeur L du transistor

(d) Récupérer W

9. Positionner les paramètres de la procédure "W(VGS,L,IDS)" pour M7
10. Appeler la procédure "W(VGS,L,IDS)" pour M7
11. Positionner les paramètres de la procédure "W(VEG,L,IDS)" pour M5
12. Appeler la procédure "W(VEG,L,IDS)" pour M5
13. Calculer les capacités parasites sur le noeud d'entrée
14. Calculer la capacité effective d'intégration
15. Calculer la puissance du signal
16. Calculer la fréquence de transition
17. Retourner les paramètres de sortie du *gmc*

Si l'on se réfère à la figure 9.15, tous les transistors n'ont pas besoin d'être individuellement dimensionnés, il suffit de dimensionner le premier transistor de chaque ligne puis d'appliquer les valeurs L et W trouvées aux autres transistors de cette ligne.

La figure 9.17 résume les échanges hiérarchiques entre les trois niveaux impliqués dans le dimensionnement électrique.

9.2.4 Simulations et résultats

Le filtre a été généré en technologie $0,13\ \mu m$ pour trois spécifications où l'on a fait varier la bande passante à rapport signal/bruit constant (OSR est aussi constant et f_s varie). Le filtre conçu doit répondre à trois principaux critères :

1. Au niveau « système » la fonction de transfert de l'intégrateur est :

$$H(s) = A_{int}/sT \tag{9.20}$$

Le circuit (figure 9.15) qui réalise cette intégration a une fonction de transfert donnée par :

$$H(s) = gm/sC \tag{9.21}$$

où gm est la transconductance des transistors M1, M11, M2, M22 et C est la capacité totale d'intégration. La fréquence de transition F_T du circuit doit donc vérifier :

$$F_T = A_{int} \cdot f_s/2 \cdot \pi \tag{9.22}$$

où f_s est la fréquence de sur-échantillonnage , égale à $1/T$, égale à $2 \cdot OSR \cdot BW$.

2. Si le critère 1 est satisfait, le bruit de circuit obtenu (somme du bruit thermique et du bruit en $\frac{1}{F}$ — figure 9.18) doit être inférieur au budget de bruit alloué.
3. Enfin, si les critères 1 et 2 sont satisfaits, il faut vérifier que la distorsion harmonique du modulateur conçu avec ce filtre est suffisamment faible.

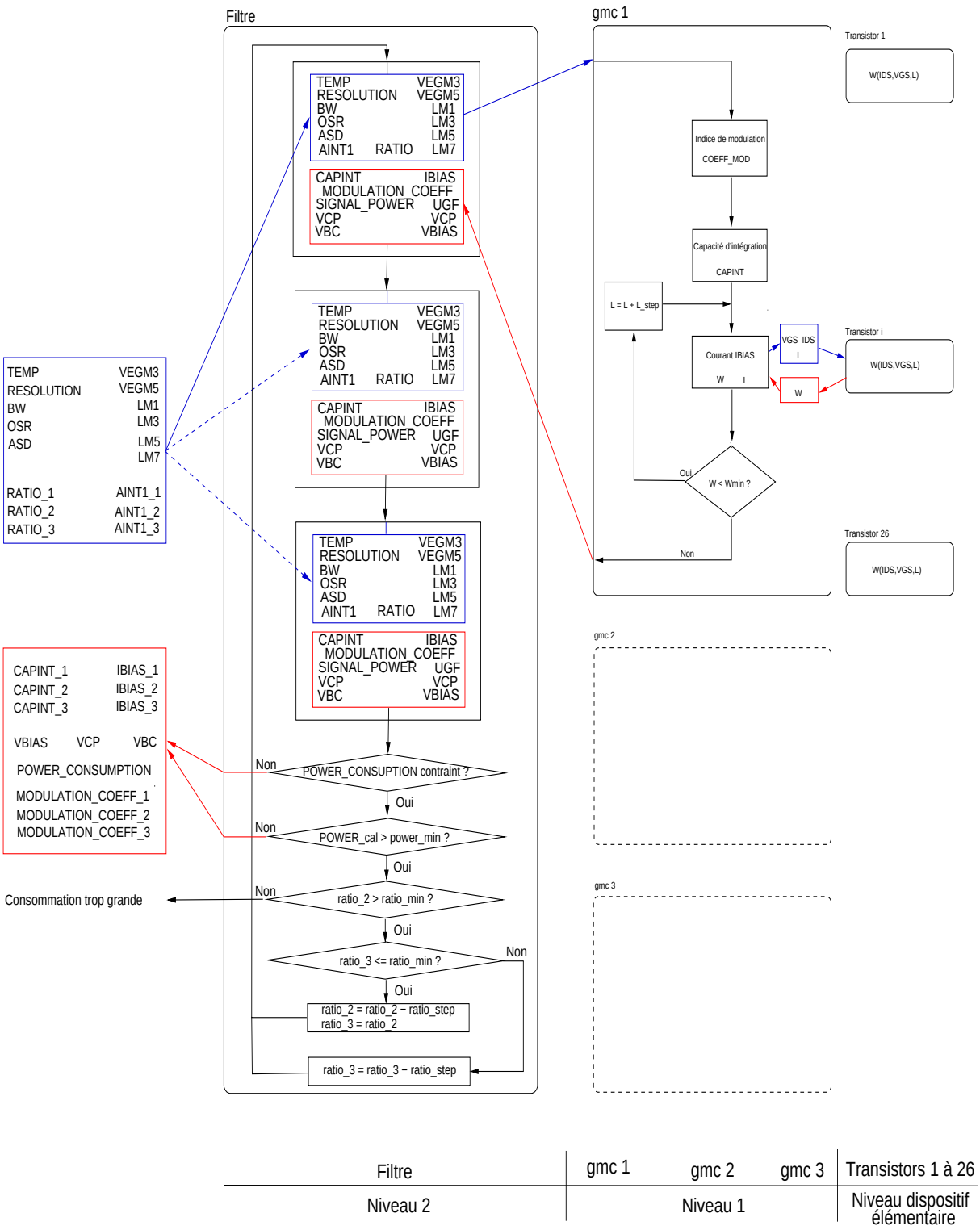


FIG. 9.17: Procédures de dimensionnement du filtre et des intégrateurs

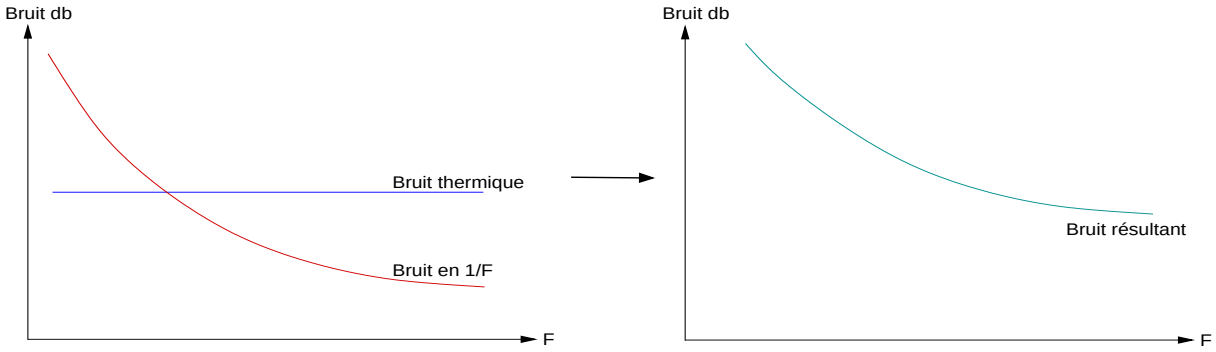


FIG. 9.18: Bruit thermique et bruit en $\frac{1}{F}$

Mesure de la fréquence de transition

La condition 1 sur le *gmc* est vérifiée en effectuant une analyse AC du premier intégrateur et en comparant la fréquence de transition calculée (équation 9.22) avec la fréquence de transition obtenue par simulation du schéma électrique généré par CAIRO+. Les paramètres d'entrée du *gmc* sont résumés dans le tableau 9.12. On observera que les transistors des deux lignes du milieu (LM3 et LM5) ont peu d'influence, les transistors des deux lignes extrêmes (LM1 et LM7) enregistrent les plus grandes variations de longueur.

TAB. 9.12: Valeurs des paramètres d'entrée du premier gmc

BW \ bit	Paramètre	8	10	11
50 kHz	TEMP (K)	300,15	300,15	300,15
	VDD (V)	0,71	0,70	0,71
	VSS (V)	0,0	0,0	0,0
	RATIO	1	1	1
	AINT1	0,0333	0,0333	0,0333
	OSR	64	64	64
	ASD	0,5	0,5	0,5
	VEGM3 (V)	0,18	0,20	0,18
	VEGM5 (V)	-0,1	-0,1	-0,1
	LM1 (μm)	7,5	8,5	8,5
	LM3 (μm)	2,5	2,5	2,5
	LM5 (μm)	2,5	2,5	2,5
	LM7 (μm)	7,5	8,5	8,5
100 kHz	TEMP (K)	300,15	300,15	300,15
	VDD (V)	0,71	0,71	0,71
Fin du tableau page suivante...				

BW \ bit	Paramètre	8	10	11
100 kHz	VSS (V)	0,0	0,0	0,0
	RATIO	1	1	1
	AINT1	0,0333	0,0333	0,0333
	OSR	64	64	64
	ASD	0,5	0,5	0,5
	VEGM3 (V)	0,1	0,18	0,18
	VEGM5 (V)	-0,1	-0,1	-0,1
	LM1 (μm)	4,5	7,5	8,5
	LM3 (μm)	2,5	2,5	2,5
	LM5 (μm)	2,5	2,5	2,5
	LM7 (μm)	4,5	7,5	8,5
500 kHz	TEMP (K)	300,15	300,15	300,15
	VDD (V)	0,71	0,71	0,71
	VSS (V)	0,0	0,0	0,0
	RATIO	1	1	1
	AINT1	0,0333	0,0333	0,0333
	OSR	64	64	64
	ASD	0,5	0,5	0,5
	VEGM3 (V)	0,20	0,18	0,1
	VEGM5 (V)	-0,1	-0,1	-0,1
	LM1 (μm)	8,5	7,5	4,5
	LM3 (μm)	2,5	2,5	2,5
	LM5 (μm)	2,5	2,5	2,5
	LM7 (μm)	8,5	7,5	4,5

Le tableau 9.13 page 130 présente l'erreur trouvée sur la fréquence de transition, l'évolution du courant de polarisation et de la capacité d'intégration en fonction de la bande passante et de la résolution. On peut remarquer que l'erreur obtenue sur la fréquence de transition est en moyenne faible (inférieure à 6 %) ; elle a tendance à augmenter avec la résolution et la bande passante.

La valeur de la capacité d'intégration est raisonnable excepté pour la résolution de 11 bits où elle pourrait commencer à poser des problèmes d'intégration.

La figure 9.19 présente l'évolution de l'erreur relative de la fréquence de transition en fonction de la résolution et de la bande passante.

La figure 9.20 présente l'évolution du courant (sous-figure 9.20(a)) et l'évolution de la capacité d'intégration (sous-figure 9.20(b)) en fonction de la résolution et de la bande passante. Sans surprise, le courant augmente suivant la résolution et la bande passante.

TAB. 9.13: Évolution de la fréquence de transition UGF, de la capacité d'intégration CAPINT et du courant IBIAS en fonction de la bande passante pour divers rapports signal/bruit pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64

BW \ bit	Paramètre	8			10			11		
50 kHz	UGF (kHz) calc mes écart %	33,92	32,00	-5,66	33,92	31,00	-8,60	33,92	32,20	-5,07
	CAPINT (F)	7,13034e-11			9,58869e-10			1,85287e-9		
	IBIAS (A)	1e-06			1,32183e-05			2,5594e-5		
	LM1 WM1 (μm)	7,5	4,934200		8,5	76,8221		7,5	123,353	
	LM3 WM3 (μm)	2,5	0,568164		2,5	6,21034		2,5	14,5736	
	LM5 WM5 (μm)	2,5	4,964990		2,5	65,2059		2,5	136,116	
	LM7 WM7 (μm)	7,5	75,75750		8,5	85,8549		7,5	75,7455	
100 kHz	UGF (kHz) calc mes err %	67,84	66,80	-1,53	67,84	64,20	-5,36	67,84	64,20	-5,36
	CAPINT (F)	3,77734e-11			4,59033e-10			1.90643e-09		
	IBIAS (A)	1e-06			1,28578e-05			5.42135e-05		
	LM1 WM1 (μm)	4,5	3,42360		7,5	62,0811		8,5	286,553	
	LM3 WM3 (μm)	2,5	1,54978		2,5	7,34655		2,5	30,0235	
	LM5 WM5 (μm)	2,5	4,95842		2,5	64,0482		2,5	289,656	
	LM7 WM7 (μm)	4,5	45,6032		7,5	75,7575		8,5	85,8336	
500 kHz	UGF (kHz) calc mes err %	339,19	311,00	-8,31	339,19	321,00	-5,36	339,19	332,00	-2,12
	CAPINT (F)	5,00497e-11			4,17707e-10			4,90771e-10		
	IBIAS (A)	8.25543e-06			6.48966e-05			6,62711e-05		
	LM1 WM1 (μm)	8,5	44,1671		7,5	312,522		4,5	220,195	
	LM3 WM3 (μm)	2,5	3,86527		2,5	35,0777		2,5	92,3474	
	LM5 WM5 (μm)	2,5	40,3191		2,5	359,658		2,5	379,527	
	LM7 WM7 (μm)	8,5	85,7693		7,5	75,7297		4,5	45,5865	

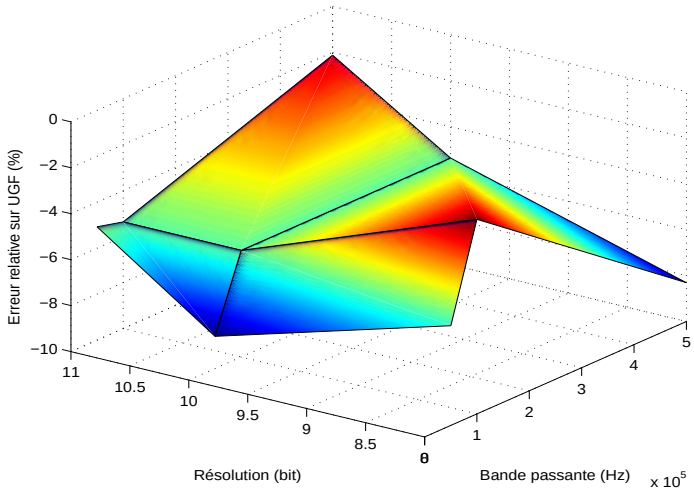
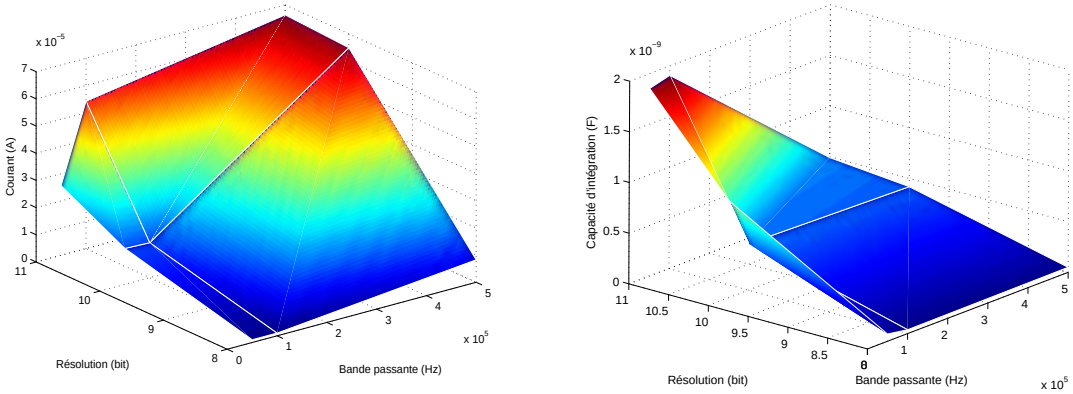


FIG. 9.19: Erreur relative sur UGF pour le premier intégrateur d'un modulateur ΣΔ du 3e ordre avec OSR=64



(a) Évolution du courant

(b) Évolution de la capacité d'intégration

FIG. 9.20: Évolution du courant et de la capacité d'intégration pour le premier intégrateur d'un modulateur ΣΔ du 3e ordre avec OSR=64

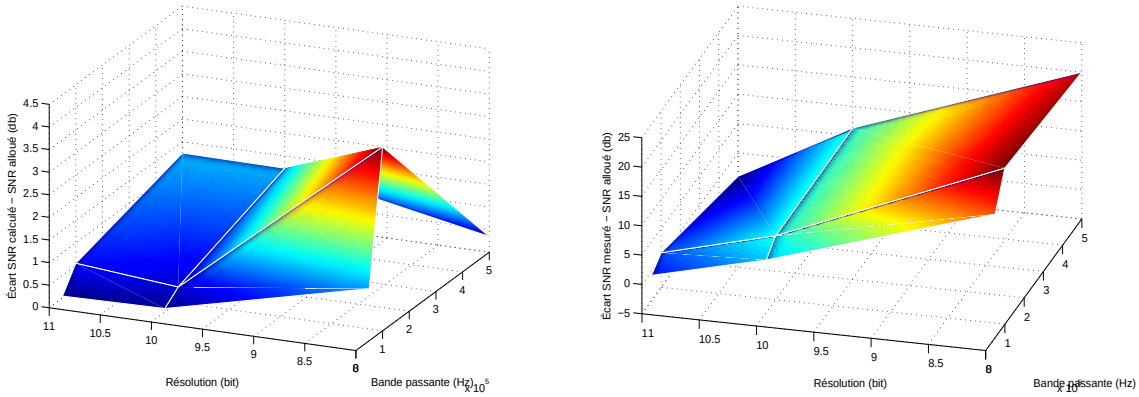
Mesure du bruit de circuit

La mesure du bruit s'effectue en régime alternatif, toujours sur le premier intégrateur. Il s'agit de relever le bruit de circuit total obtenu par simulation (bruit thermique avec le bruit en $\frac{1}{F}$, noté Bruit circuit) et de calculer le rapport signal sur bruit $SNR_{Th+\frac{1}{F}}$ défini par :

$$SNR_{Th+\frac{1}{F}db} = 10 \cdot log_{10}\left(\frac{SIGNAL\_POWER}{(\text{Bruit circuit})^2}\right) \tag{9.23}$$

Ce rapport signal/bruit est comparé avec le budget de bruit résultant de la résolution imposée (on a approximativement $SNR_{db} = \text{RESOLUTION}_{bit} \cdot 6$). Si la différence est suffisamment faible, une vérification fonctionnelle en régime transitoire du modulateur est alors entreprise afin de contrôler la densité spectrale et de mesurer cette fois-ci le bruit de quantification noté Q_{mes} (on calcule alors la puissance du bruit sur le $\Sigma\Delta$ complet par une analyse de Fourrier. Elle est donnée sans le bruit thermique ni sans le bruit en $\frac{1}{F}$ car le simulateur électrique dont nous disposons ne permet pas, actuellement, une telle mesure) .

Le tableau 9.14 page 133 et la figure 9.21 présentent l'évolution de l'écart sur le rapport signal/bruit. L'écart entre le $SNR_{Th+\frac{1}{F}}$ calculé et le SNR_{Budget} alloué est toujours positif (sous-figure 9.21(a)), tandis que l'écart entre le SNR_{mes} mesuré et le SNR alloué (sous-figure 9.21(b)) montre que la résolution 11 bits pour la bande passante 500 kHz ne permet pas de tenir les spécifications. Il s'agit vraisemblablement de la limite de réalisation de l'intégrateur pour la technologie cible et l'architecture retenue, mais il faut aussi considérer que le dimensionnement électrique a été effectué sans tenir compte du bruit en $\frac{1}{F}$ des transistors (le modèle électrique implanté n'inclut pas cette donnée).



(a) Écart SNR calculé - SNR alloué

(b) Écart SNR mesuré - SNR alloué

FIG. 9.21: Écart en db entre $SNR_{Th+\frac{1}{F}}$ et $SNR_{Q_{mes}}$ et le SNR alloué pour le premier intégrateur d'un modulateur $\Sigma\Delta$ du 3e ordre avec OSR=64

Simulation transitoire pour RESOLUTION=11 bits et BW=100 kHz

Une simulation transitoire a été réalisée pour un modulateur dont la résolution est de 11 bits et la bande passante de 100 kHz (l'écart entre le SNR calculé et le SNR alloué pour cette spécification est de +0,55 db). Les autres éléments (comparateur, convertisseurs numérique-analogique...) sont idéaux. Les valeurs des paramètres d'entrée du filtre pour cette spécification sont résumés dans le tableaux 9.15.

TAB. 9.14: Rapports SNR en db résultant de la précision demandée ($Budget$), du bruit de circuit calculé ($SNR_{Th+\frac{1}{F}}$) et du bruit de quantification mesuré (Q_{mes}) pour un modulateur ΣΔ du 3e ordre avec Vdd=0,71 V, OSR=64

BW \ Résolution	Paramètre	8			10			11		
50 kHz	SIGNAL_POWER (W)	2,85323e-14			2,47503e-12			1,84397e-11		
	Bruit circuit(A)	5,890668E-10			1,558523E-09			2,142667E-09		
	SNR (db)									
	$Budget$ $Th+\frac{1}{F}$ Q_{mes}	48	49,15	64,00	60	60,08	64,00	66	66,04	65,3
100 kHz	SIGNAL_POWER (W)	1.10793e-13			4,65387e-12			7,62331e-11		
	Bruit circuit(A)	8,297070E-10			2,074760E-09			4,076476E-09		
	SNR (db)									
	$Budget$ $Th+\frac{1}{F}$ Q_{mes}	48	52,07	69,60	60	60,34	65,90	66	66,55	66,9
500 kHz	SIGNAL_POWER (W)	9,2647e-13			1,16966e-10			5,22465e-10		
	Bruit circuit(A)	3,678868E-09			9,383771E-09			9,915942E-09		
	SNR (db)									
	$Budget$ $Th+\frac{1}{F}$ Q_{mes}	48	48,35	68,00	60	61,23	66.30	66	67,25	62,00

TAB. 9.15: Valeur des paramètres d'entrée du filtre

Paramètre d'entrée	Valeur
TEMP (kelvin)	300,15
VDD (V)	0,71
VSS (V)	0,0
RATIO_1	1
RATIO_2	1
RATIO_2	1
RESOLUTION	11
BW (kHz)	100
AINT1_1	0,0333
AINT1_2	0,5
AINT1_3	0,5
OSR	64
ASD	0,5
VEGM3 (V)	0,18
VEGM5 (V)	-0,1
LM1 (μm)	8,5
LM3 (μm)	2,5
LM5 (μm)	2,5
LM7 (μm)	8,5

Le résultat de la transformée de Fourrier, conforme aux attentes, est présenté en figure 9.22. Le rapport signal/bruit mesuré est de 66,9 db, soit 0,9 db au-delà des 66 db exigés par la résolution de 11 bits. Le filtre satisfait les contraintes imposées au début de la synthèse et le bruit a bien été rejeté dans les fréquences non utiles. La distorsion harmonique est peu importante.

La figure 9.23 présente le dessin des masques du filtre généré (sans le routage). On peut remarquer pour chaque intégrateur que les deux transistors d'alimentation M5bias et M7bias ne sont pas alignés avec respectivement la première ligne et la deuxième ligne. L'ajout d'un transistor fictif dans le même conteneur pourrait concourir à symétriser le dessin des masques.

Simulation transitoire pour RESOLUTION=8 bits et BW=50 kHz

L'écart sur le SNR pour cette spécification est cette fois-ci de +1,15 db. La spécification est plus facilement atteinte, le spectre présente moins d'harmoniques (figure 9.24).

Le rapport signal/bruit mesuré est de 64,0 db, soit 16,0 db au-delà des 48 db exigés par la résolution de 8 bits. Le filtre satisfait donc les contraintes imposée au début de la synthèse.

La figure 9.25 présente le dessin des masques du filtre généré (sans le routage).

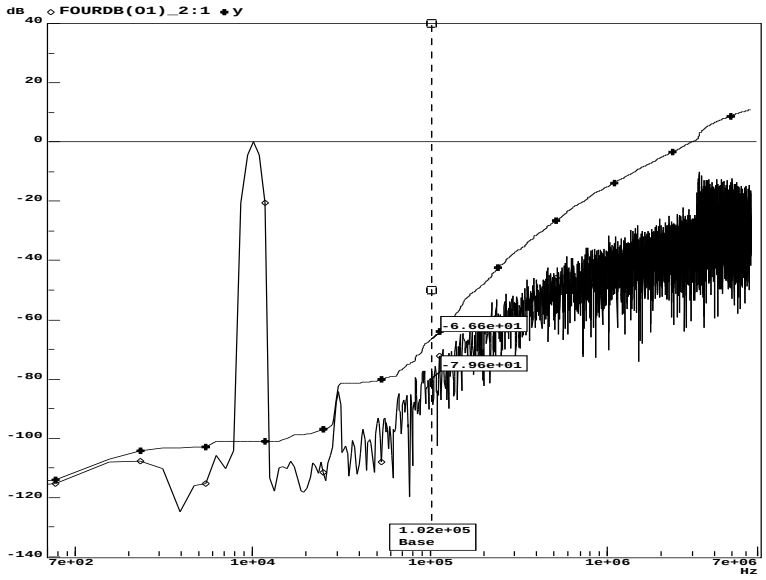


FIG. 9.22: Mesure de la densité spectrale pour RESOLUTION=11 bits et BW=100 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64

Les temps de calculs varient de quelques secondes pour une faible bande passante et une faible résolution, à dix minutes pour la spécification la plus sévère.

9.3 Conclusion

La conception d'un générateur demande un investissement certain. Le bénéfice que l'on retire est la possibilité, une fois le générateur achevé, de pouvoir caractériser une topologie en évaluant rapidement la gamme de performances atteintes pour une technologie donnée et une topologie donnée.

La méthodologie de CAIRO+ a été appliquée avec succès dans ce chapitre lors de la réalisation d'un générateur d'OTA, qui a été validé pour deux technologies différentes et deux spécifications différentes, et lors de la réalisation d'un générateur de filtre pour modulateur $\Sigma\Delta$. Le temps investi dans la conception (plusieurs mois) est largement compensé par la rapidité de génération d'un schéma dimensionné et d'un dessin des masques (quelques minutes), permettant ainsi de multiplier les évaluations de performances. Les bénéfices attendus sont encore accrus par le fait que les générateurs sont réutilisables dans différents contextes.

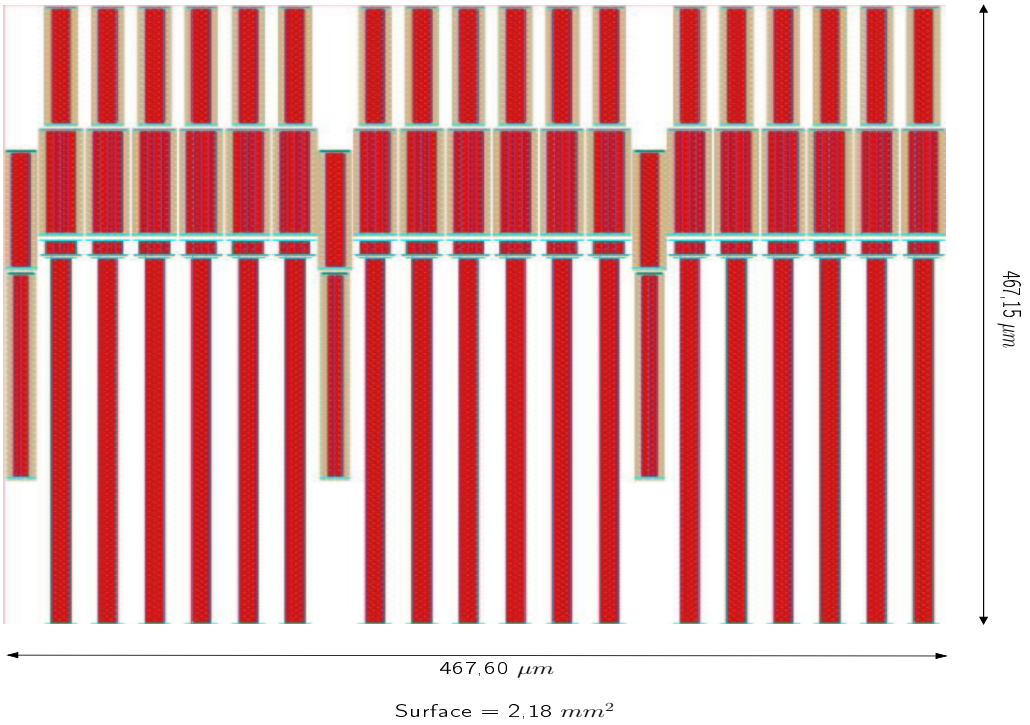


FIG. 9.23: Dessin des masques du filtre pour RESOLUTION=11 bits et BW=100 kHz pour un modulateur $\Sigma\Delta$ du 3^e ordre avec Vdd=0,71 V, OSR=64

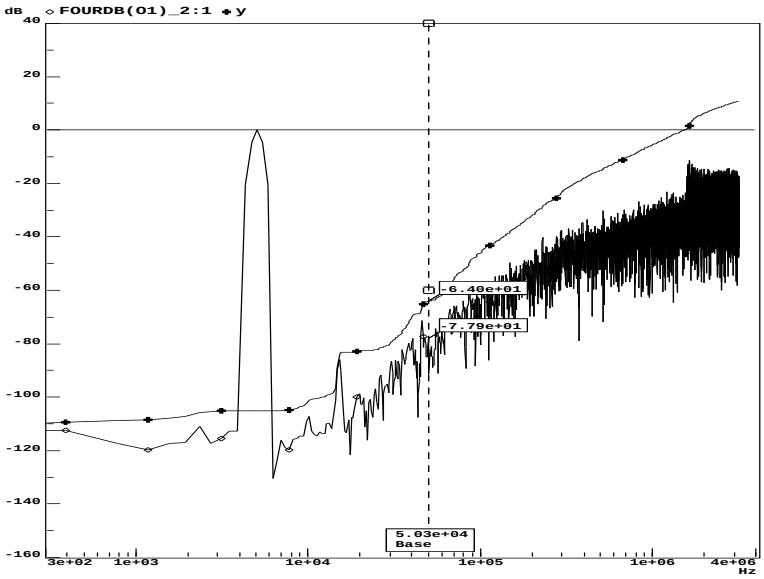


FIG. 9.24: Mesure de la densité spectrale pour RESOLUTION=8 bits et BW=50 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64

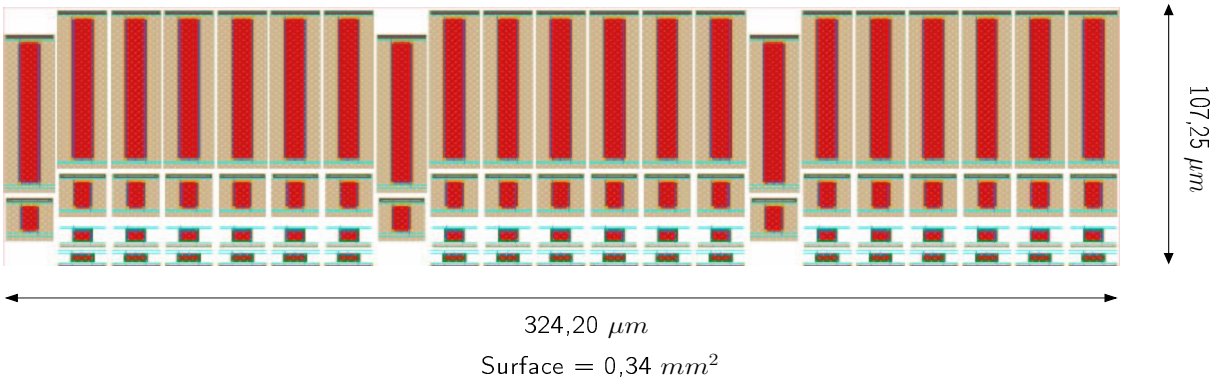


FIG. 9.25: Dessin des masques du filtre pour RESOLUTION=8 bits et BW=50 kHz pour un modulateur $\Sigma\Delta$ du 3e ordre avec Vdd=0,71 V, OSR=64

Chapitre 10

Conclusion

La conception de circuits analogiques est arrivée à un tournant de son développement : face à l'accélération de la mise sur le marché de nouveaux procédés technologiques, face à la pression commerciale exercée, mais aussi face au manque d'outils de synthèse dédiés et au manque d'algorithmes permettant une automatisation de la conception, il est impératif qu'une évolution ait lieu. Elle touche tout autant le côté technique que le facteur humain de la conception analogique de part la mise en place de plateformes de développement reposant sur de nouvelles méthodologies, et de part l'acceptation de ces dernières.

Cette thèse s'est attachée à dégager les principes invariants de la conception analogique en formalisant les points essentiels pour les regrouper dans un environnement contrôlé permettant à la fois la capitalisation rationnelle des connaissances et leur réutilisation dans de multiples contextes. La méthodologie mise en œuvre brise le schéma classique conception non hiérarchique-extraction-simulation par l'utilisation de générateurs impliquant une interaction forte entre le dimensionnement électrique et le dessin des masques.

Des générateurs communicants

L'organisation des données est régie par trois entités essentielles : un patron électrique pour le schéma non dimensionné, un patron physique pour le dessin des masques et une interface fonctionnelle pour le dimensionnement électrique. La conception imposée est dès le début hiérarchique grâce à la disponibilité de bibliothèques. S'il doit penser en termes de réutilisation, l'analogicien n'est pas dépossédé du contrôle du circuit : outre le choix de la topologie, le placement est dirigé par un placement relatif qu'il maîtrise.

Le générateur est un composant actif grâce à l'interface fonctionnelle : le concepteur analogicien pense son circuit en terme de dimensionnement électrique (la finalité) et en terme de services rendus (la réutilisation). Il écrit des procédures de dimensionnement qui ont la propriété d'instaurer un dialogue hiérarchique pour résoudre un problème donné. Le dimensionnement du circuit électrique est obtenu par raffinement en s'aidant des procédures disponibles dans les générateurs instanciés et en mettant en place les procédures qui rendront elles-mêmes un service aux niveaux supérieurs qui auront instancié ce générateur. L'écriture de l'interface fonctionnelle est encadrée de manière à aider la formalisation.

La communication hiérarchique permet de réaliser l'exploration du domaine de conception. Le dialogue établi entre les instances des dispositifs utilisés est une recherche de compromis entre les spécifications électriques réalisables par les fils et les contraintes du module père. Ce dialogue doit aboutir à une solution réalisable ou à une modification des spécifications, voire à un changement de topologie dans les cas extrêmes.

Une interaction physique-électrique forte

La portabilité fonctionnelle et technologique est pensée dès la conception d'un générateur. Les phases d'extraction et de simulation ont été rejetées en dehors de la boucle de conception grâce aux modèles électriques disponibles sous forme de bibliothèques pour le niveau le plus bas de la hiérarchie, et grâce aux procédures de dimensionnement qui ont le même rôle pour les niveaux supérieurs. Les effets parasites induits par le dessin des masques sont estimés pour être pris en compte au plus tôt.

Les contraintes purement analogiques du dessin des masques des dispositifs sont respectées pour produire un dessin des masques portable, de qualité, dont les propriétés sont reproductibles tout en fournissant les informations nécessaires à la phase de dimensionnement électrique.

La capitalisation des connaissances

Les deux points précédents tendent à guider le concepteur analogicien dans la formalisation de ses connaissances. Chaque équation, chaque astuce de réalisation ou tour de main trouve sa place dans l'une ou l'autre des parties constitutives d'un générateur, en ayant été ordonnés et exprimés de façon explicite. Il est évident qu'un effort d'abstraction est nécessaire, il est même imposé (en douceur) par la méthodologie mise en place.

Une plateforme de développement

La méthodologie exposée dans cette thèse a donné lieu à la création d'une plateforme de développement et d'un langage d'aide à la création de générateurs utilisant les structures de données de la chaîne de *CAO* Alliance [Ga92]. Elle représente plus de quatre-vingt dix mille lignes de code C++ (et accessoirement la résolution d'une bogue du compilateur *gcc*) et a été compilée pour les plateformes Linux, Solaris et IRIX. Les composants suivants sont disponibles au concepteur analogicien :

- le langage CAIRO+ ;
- une bibliothèque de dispositifs élémentaires (libMOS) ;
- une bibliothèque de modules (libOTA), un intégrateur de courant (*gmc* et un filtre passe-bas) ;
- une bibliothèque de motifs pour le dessin des masques des transistors (libCpPattern) ;
- une bibliothèque contenant un langage procédural de manipulation de dessin des masques réel qui a servi à la création des motifs (libCpRmf) ;
- une bibliothèque pour le routage procédural (libCpPgf) ;
- deux bibliothèques contenant les modèles électriques des transistors et les fonctions d'accès (libcairoBSIM3V3 et libcairoMOS).

Hormis les bibliothèques de modèles électriques de transistors, tous les autres éléments sont issus de cette thèse. D'autres bibliothèques de dispositifs élémentaires et de modules tiers sont en préparation.

Deux exemples ont servi de démonstrateurs de la méthodologie : un OTA simple et un intégrateur gm/c. Les résultats obtenus montrent la validité de l'approche.

Perspectives

CAIRO+ doit encore se développer pour séduire les concepteurs analogiciens et leur montrer une nouvelle voie :

- Les contraintes fortes de symétries du dessin des masques sont prises en compte dans la conception des dispositifs élémentaires. Les contraintes physiques entre les modules (par exemple un nombre de repliement identique pour les transistors de deux modules différents et appartenant au même niveau hiérarchique — ayant un père direct commun) ne sont en revanche pas supportées. Après une communication hiérarchique qui permet de transmettre verticalement de contraintes, il reste à développer une communication transversale.
- Le routage inter-modules est procédural. Un routeur physique prenant en compte les contraintes analogiques déchargerait le concepteur d'une tâche contraignante. Il devra tout de même accepter un routage dirigé où l'analogicien aurait imposé le routage global de certains fils critiques.
- L'extension de la bibliothèque de dispositifs élémentaires est essentielle. Il faut aussi proposer des circuits prêts à l'emploi (exécutable) et des modules réutilisables (générateur).

Évolutions

Le langage CAIRO+ est le fondement de la méthodologie de CAIRO+. La tendance actuelle est d'améliorer et de faciliter le travail de conception par l'introduction d'une interface graphique mutualisant les ressources disponibles. Une version graphique du langage CAIRO+ sur le modèle UML libérerait le concepteur analogicien des tâches répétitives que sont l'écriture de l'interface électrique, la description du schéma non dimensionné électrique et du placement relatif pour se consacrer à la mise au point de procédures de dimensionnement, mise au point facilitée par l'introduction d'éléments préconçus. Introduire un méta-language au dessus de CAIRO+ autoriserait des vérifications qu'une exécution seule du code ne permet pas, une meilleur gestion des interactions entre les générateurs en proposant automatiquement un canevas à compléter, sans compter une documentation automatique.

*
* *

Annexe A

Plateforme de tests l'OTA

Ce paragraphe présente le circuit utilisé en simulation *spice* pour caractériser l'OTA de type P généré par CAIRO+. Le schéma électrique dimensionné devra par la suite être confronté aux simulations issues du schéma électrique dimensionné extrait du dessin des masques.

A.1 Mesure de la tension de décalage

Le chemin constitué par les deux branches différentielles de l'OTA n'est pas totalement symétrique en raison de la présence du miroir de courant dont l'un des drains est relié aux grilles. Cela engendre un décalage de tension en sortie V_{OUT} par rapport à la tension de mode commun prévue initialement, décalage qu'il est possible de mesurer et de compenser.

Lorsque deux tensions identiques sont appliquées aux entrées *inp* (V_{IN+}) et *inm* (V_{IN-}), la tension de sortie présente un décalage par rapport à la tension prévue. La mesure de ce décalage est présentée à l'une des entrées différentielles pour compenser la disymétrie (figure A.1).

- La tension V_{sc} (*shortcut*) est nulle. Il s'agit d'un artifice permettant d'effectuer des mesures en *spice* en plaçant une sonde entre les nœuds *inp* (*input plus*) et *ocm* (*output common mode*). La tension mesurée doit être égale à $\frac{V_{DD}}{2}$ (le fonctionnement de l'OTA est centré sur $\frac{V_{DD}}{2}$).
- V_{icm} est la tension de mode commun en entrée (*input common mode*). Elle est choisie égale à $\frac{V_{DD}}{2}$ (le fonctionnement de l'OTA est centré sur $\frac{V_{DD}}{2}$).
- V_{p1} est la tension de polarisation déterminée par CAIRO+.
- La tension V_{iov} est la tension de décalage à appliquer pour que la tension de sortie soit égale à $\frac{V_{DD}}{2}$. C'est donc cette tension que l'on fait varier pour compenser la dissymétrie.

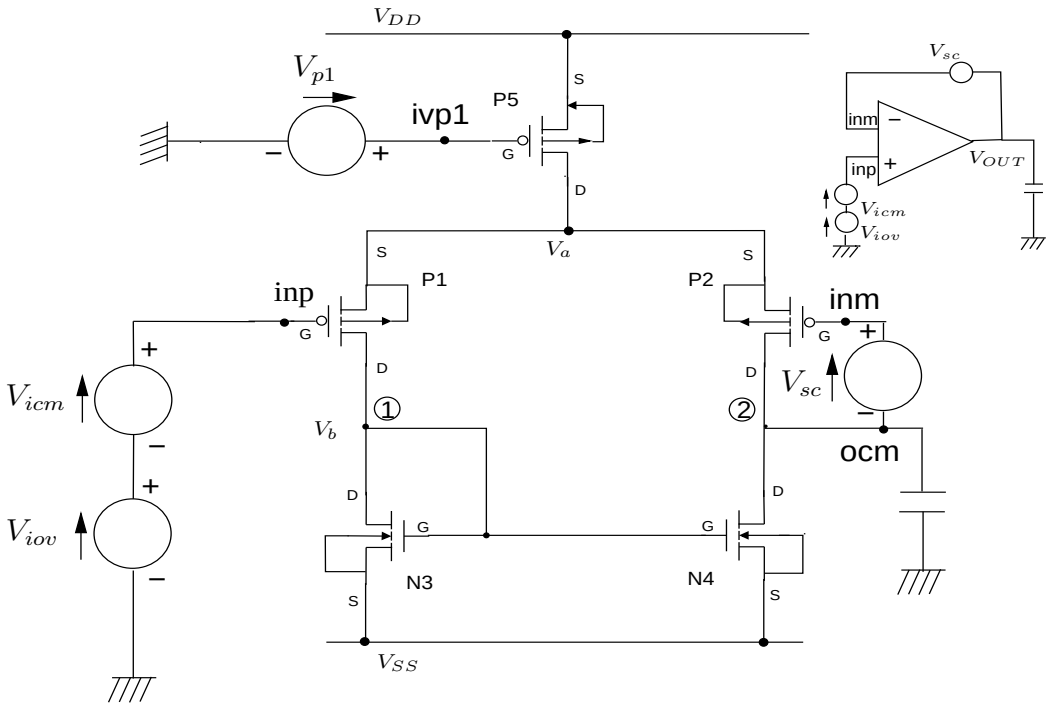


FIG. A.1: OTA de type P — Mesure de la tension de décalage

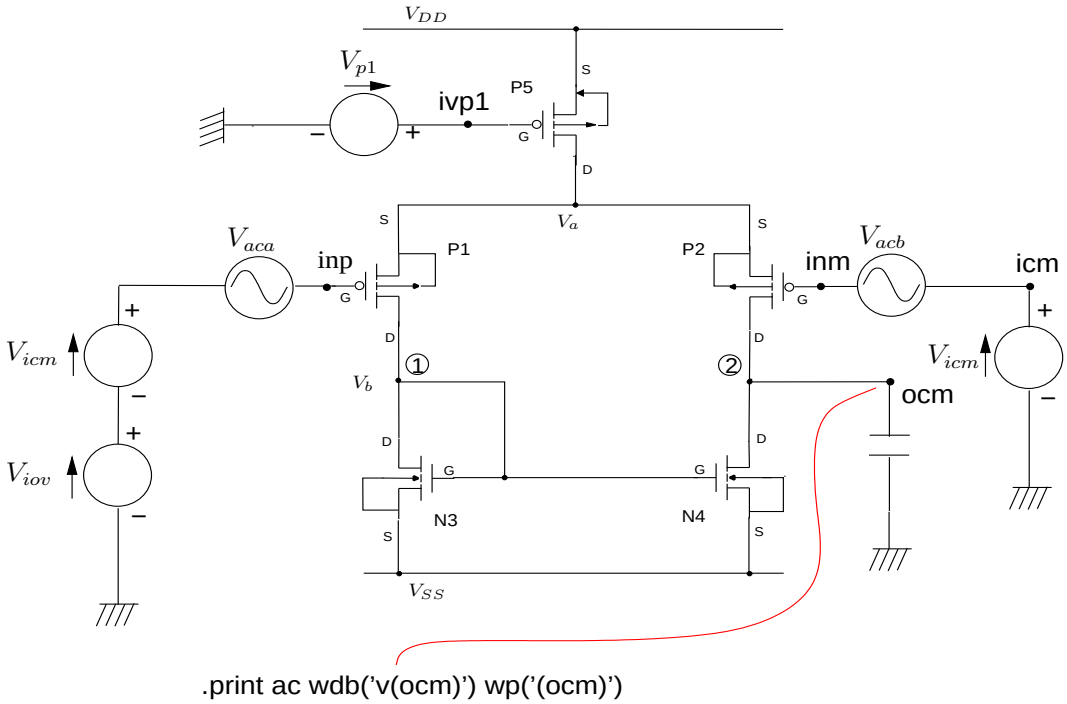


FIG. A.2: OTA de type P — Analyse petit signal

A.2 Analyse petit signal

Le dispositif utilisé lors de l'analyse petit signal est montré en figure A.2. Les tensions V_{icm} et V_{p1} sont identiques à celles utilisées pour mesurer le tension de décalage.

- Le nœud *inp* est au potentiel $V_{aca} + V_{icm} + V_{iov}$ où V_{aca} est une tension variable opposée à V_{acb}, V_{icm} est la tension de mode commun en entrée et V_{iov} est la tension de décalage (*input offset voltage*).
- Le nœud *inm* est au potentiel $V_{acb} + V_{icm}$.
- V_{p1} est la tension de polarisation déterminée par CAIRO+.

On effectue une analyse en régime alternatif ($sin(\omega \dot{t})$) de la phase et de la tension au nœud *ocm* pour une fréquence variant de 0 à 160 MHz.

Annexe B

Mode d'emploi de libMOS

```
-------------------------------------------------------------------------------
| Toute copie, reproduction, meme partielle est interdite sans le consentement  |
| express de l'auteur. Toute modification lors de la reproduction est interdite |
-------------------------------------------------------------------------------

Redacteur        Pierre Nguyen Tuong
Version initiale 1.0  20/03/2003 CAIRO+ version 3.3.0  Drewry's Bluff
Revision         1.1  28/01/2004 CAIRO+ version 5.1.0  Sutherland's Station
Revision         1.2  30/01/2004 CAIRO+ version 6.1.4  Kolb's Farm
Revision         1.3  03/02/2004 CAIRO+ version 6.1.4  Kolb's Farm
Revision         2.0  29/10/2004 CAIRO+ version 7.0.0  Parker's Cross Roads
Revision         2.1  06/01/2005 CAIRO+ version 8.1.0  Front Royal
Revision         2.1  09/02/2005 CAIRO+ version 9.0.0  Guard Hill
Revision         3.0  03/08/2005 CAIRO+ version 10.0.0 Honey Springs
Revision         3.0  03/08/2005 CAIRO+ version 11.0.1 LaFourche Crossing

                        ---------------------------------
                        | Generateurs          | Version |
-------------------------------------------------|---------|
| Bibliotheque  | libMOS  |                     |         |
|-------------------------| TR_MOS DP_MOS CM_MOS |         |
| Technologie   | MOS     |                     |         |
|=================================================| CAIRO+  |
| Inventeurs    | License | Copyright           |         |
|-------------------------------------------------|         |
|               |         |                     |         |
| PNT LdL       | QPL     | ASIM                |         |
|               |         |                     |         |
-------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------------------
Description: Transistor simple replie
-------------------------------------------------------------------------------------------------------------
Generateur TR_MOS
-------------------------------------------------------------------------------------------------------------

Schema electrique
-----------------

                                 drain
                                |
                              __|
                             ||
                     gate ---||----- bulk (optionnel)
                             ||__
                                |
                                |
                                  source

Creation
--------

  CAIRO_CREATE(char *libName,char *genName,char *name,char *trName,char typebool,bool bulkToSource)

  libName      = "libMOS"
  genName      = "TR_MOS"
  name         = nom d'instance
  trName       = nom du transistor
```

```
  type         = TRANSN ou TRANSP
  bulkToSource = true ou false (source reliee en interne au bulk si bulkToSource = true. Dans ce cas pas de connecteur bulk.)


  Ordre des connecteurs pour l'instanciation:

  Si bulkToSource = false   :    1 drain
                                 2 gate
                                 3 source
                                 4 bulk

  Si bulkToSource = true    :    1 drain
                                 2 gate
                                 3 source

Modes de realisation
--------------------

* Parametres

  ring         = true ou false (presence d'un anneau de garde si ring = true)
  dummy        = true ou false (presence de transistors fictifs sur les bords si dummy = true)

* Arborescence:

BEGIN_CIRCUIT nom_circuit
  INSTANCE nom_instance (dummy,false) (ring,false)
END_CIRCUIT


Plan de masse
-------------

  Les connecteurs physiques sont a l'est et a l'ouest (acces par les index)

                          ------------------------------------------------------------------------------
                          | Nom des connecteurs | Nom des connecteurs                      | Index | Face  |
                          |     electriques     | physiques correspondant                  |       |       |
                          |----------------------------------------------------------------------------|
                          | source              | source                                   | 0     | ouest |
                          |                     |                                          | 1     | est   |
                          |----------------------------------------------------------------------------|
                          | gate                | gate                                     | 0     | ouest |
                          |                     |                                          | 1     | est   |
                          |----------------------------------------------------------------------------|
                          | drain               | drain                                    | 0     | ouest |
                          |                     |                                          | 1     | est   |
  ------------------------------------------------------------------------------------------------------|
  | si bulkToSource = false | bulk                | bulk                                     | 0     | sud   |
  |                         |                     |                                          | 1     | sud   |
  |                         |                     |                          ----------------|-------|-------|
  |                         |                     |                          | si ring = true  | 2     | nord  |
  |                         |                     |                          |                 | 3     | nord  |
  |                         |                     |                          |                 | 4     | ouest |
  |                         |                     |                          |                 | 5     | est   |
  ------------------------------------------------------------------------------------------------------


                            bulk       bulk
                           (bulk.2)   (bulk.3)
                              |          |
                           ___|__________|____
                          |                   |
    (source.0)   source --|-------------------|-- source   (source.1)
    (gate.0)     gate   --|-------------------|-- gate     (gate.1)
                          |                   |
                          |                   |
    (bulk.4 )    bulk   --|-------------------|-- bulk     (bulk.5)
                          |                   |
                          |                   |
    (drain.0)    drain  --|-------------------|-- drain    (drain.1)
                          |                   |
                          |___________________|
                              |          |
                              |          |
                            bulk       bulk
                           (bulk.0)   (bulk.1)


Parametres
----------

                    ---------------------------------------------------------------------------------------------
                    | Parametre | Type     | Classe       | Type C++ | Unite   | Valeur   | Statut possible     |
                    |           |          |              |          |         | nominale |                     |
  ------------------------------------------------------------------------------------------------------------|
  |                 | TEMP      | CP_IN    | CP_SECONDARY | double   | Kelvin  | 300.15 K | -                   |
  | Entrees         | VDS       | CP_IN    | CP_SECONDARY | double   | Volt    | techno   | -                   |
```

	VBS	CP_IN	CP_SECONDARY	double	Volt	techno	-
	VEG	CP_IN	CP_SECONDARY	double	Volt	techno	-
	NFINGER	CP_IN	CP_SECONDARY	long	-	-	-
	VTH	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID
							CP_NAV
	VDSAT	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID
							CP_NAV
	CGD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CGS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CGB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
Sorties	CBD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CSD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CM	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CMB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CMX	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CBS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	MODE	CP_OUT	CP_SECONDARY	string	---	techno	"cut off"
							"subthreshold"
							"strong inversion & linear"
							"strong inversion & saturation"
							"weak inversion & linear"
							"weak inversion & saturation"
							"Unknown mode"
	W	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV
							CP_VALID
Entrees-sorties							CP_TOO_SMALL
	L	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV
							CP_VALID
							CP_TOO_SMALL
	IDS	CP_INOUT	CP_SECONDARY	double	Ampere	techno	CP_VALID
							CP_TOO_LARGE
		CP_OUT	CP_SECONDARY	double	Ampere	techno	CP_VALID
							CP_TOO_LARGE
	VGS	CP_IN	CP_SECONDARY	double	Volt	techno	-
Mutable		CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID
							CP_NAV
	GDS	CP_IN	CP_SECONDARY	double	Siemens	techno	-
		CP_OUT	CP_SECONDARY	double	Siemens	techno	CP_VALID
							CP_NAV
	GM	CP_IN	CP_SECONDARY	double	Siemens	techno	-
		CP_OUT	CP_SECONDARY	double	Siemens	techno	CP_VALID
							CP_NAV

VDD et VSS en variables globales CAIRO+

Contraintes

Parametre	Contraintes
L	L >= Lmin (contrainte interne)
W	W >= Wmin (contrainte interne)

DSES

Parametres fournis en entree	Parametres calcules	Nom de procedure	Classe	Etat possibles
:	VTH VDSAT IDS			
:				
:	GM GDS			
:				
:	CGS CGB CGD CBD CSD CM			
: VGS W L	CMB CMX CBS	IDS(VGS,W,L)	CP_UNCOMPLETE	CP_OK
:				
:	MODE			
:				

			L			
						CP_OK
			VTH VDSAT IDS			1
						2
			GM GDS			3
Groupe 1						
VGS	TEMP VDS VBS	VGS W IDS	CGS CGB CGD CBD CSD CM	L(VGS,W,IDS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			W			
						CP_OK
			VTH VDSAT IDS			4
						5
			GM GDS			6
		VGS L IDS	CGS CGB CGD CBD CSD CM	W(VGS,L,IDS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			VTH	VTH(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			VTH VDSAT	VDSAT(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			VTH VDSAT MODE	MODE(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			GM	GM(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			GDS	GDS(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			CGS CGB CGD CBD CSD CM	CAPA(VGS,W,L)	CP_UNCOMPLETE	CP_OK
		VGS W L	CMB CMX CBS			
			VTH VDSAT IDS			
			GM GDS			
			CGS CGB CGD CBD CSD CM			
			CMB CMX CBS	ALL(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			MODE			
			VDSAT IDS			
			VTH VGS			
			GM GDS			
		VEG W L	CGS CGB CGD CBD CSD CM	IDS(VEG,W,L)	CP_UNCOMPLETE	CP_OK
			CMB CMX CBS			
			MODE			
			L			
						CP_OK
			VDSAT VTH VGS IDS			7
						8
Groupe 2	TEMP VDS VBS		GM GDS			9
VEG						
		VEG W IDS	CGS CGB CGD CBD CSD CM	L(VEG,W,IDS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			W			
						CP_OK
			VDSAT VTH VGS IDS			10
						11
			GM GDS			12
		VEG L IDS	CGS CGB CGD CBD CSD CM	W(VEG,L,IDS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			L			
						CP_OK
			VTH VDSAT IDS			1
						2
			GM			3
		VGS GDS W	CGS CGB CGD CBD CSD CM	L(VGS,W,GDS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
Groupe 3	TEMP VDS VBS					

		:	W				
		:				CP_OK	
		:	VTH VDSAT IDS			4	
		:				5	
		:	GDS			6	
		:					
		: VGS GM L	CGS CGB CGD CBD CSD CM	W(VGS,L,GM)	CP_UNCOMPLETE		
		:	CMB CMX CBS				
		:					
		:	MODE				
Directe	W	L	W L [NFINGER]	direct_size	CP_COMPLETE	13 14	

Note: NFINGER est un parametre optionnel permettant de fixer le nombre de repliements

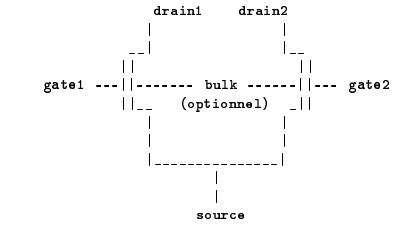
Messages

Etat de retour	Message
1	"L(VGS,W,IDS) : IDS is too large, try to decrease current"
2	"L(VGS,W,IDS) : IDS is too small, try to increase current"
3	"L(VGS,W,IDS) : L is too small for the current, try to increase IDS"
4	"W(VGS,L,IDS) : IDS is too large, try to decrease current"
5	"W(VGS,L,IDS) : IDS is too small, try to increase current"
6	"W(VGS,L,IDS) : W is too small for the current, try to increase IDS"
7	"L(VEG,W,IDS) : IDS is too large, try to decrease current"
8	"L(VEG,W,IDS) : IDS is too small, try to increase current"
9	"L(VEG,W,IDS) : L is too small for the current, try to increase IDS"
10	"W(VEG,L,IDS) : IDS is too large, try to decrease current"
11	"W(VEG,L,IDS) : IDS is too small, try to increase current"
12	"W(VEG,L,IDS) : W is too small for the current, try to increase IDS"
13	"direct_size : W is too small - DRM rules"
14	"direct_size : L is too small - DRM rules"

Description: Paire differentielle

Generateur DP_MOS

Schema electrique



Creation

CAIRO_CREATE(char *libName,char *genName,char *name,char *trName1,char *trName2,char type,bool bulkToSource)

libName = "libMOS"
genName = "DP_MOS"
name = nom d'instance
trName1 = nom du transistor 1
trName2 = nom du transistor 2
type = TRANSN ou TRANSP
bulkToSource = true ou false (source reliee en interne au bulk si bulkToSource = true. Dans ce cas pas de connecteur bulk.)

Ordre des connecteurs pour l'instanciation:

Si bulkToSource = false :
1 drain1
2 drain2
3 gate1
4 gate2
5 source
6 bulk

```
  Si bulkToSource = true    :    1 drain1
                                 2 drain2
                                 3 gate1
                                 4 gate2
                                 5 source


Modes de realisation
--------------------

* Parametres

  ring         = true ou false (presence d'un anneau de garde si ring = true)
  dummy        = true ou false (presence de transistors fictifs sur les bords si dummy = true)
  stack        = -1 ou > 1 (nombre d'empilements. -1 vaut pour un empilement libre, sinon vaut
                 le nombre d'empilements desire)

* Arborescence:

BEGIN_CIRCUIT nom_circuit
  INSTANCE nom_instance (dummy,false) (ring,false) (stack,-1)
END_CIRCUIT


Plan de masse
-------------

  Les connecteurs physiques sont au nord et au sud (acces par les index)

                          ------------------------------------------------------------------------
                          | Nom des connecteurs | Nom des connecteurs                  | Index | Face  |
                          |     electriques     | physiques correspondant              |       |       |
                          |----------------------------------------------------------------------|
                          | source              | source                               | 0     | sud   |
                          |                     |                                      | 1     | nord  |
                          |----------------------------------------------------------------------|
                          | gate1               | gate1                                | 0     | sud   |
                          |                     |                                      | 1     | nord  |
                          |----------------------------------------------------------------------|
                          | gate2               | gate2                                | 0     | sud   |
                          |                     |                                      | 1     | nord  |
                          |----------------------------------------------------------------------|
                          | drain1              | drain1                               | 0     | sud   |
                          |  |                  |                                      | 1     | nord  |
                          |----------------------------------------------------------------------|
                          | drain2              | drain2                               | 0     | sud   |
                          |                     |                                      | 1     | nord  |
  ------------------------------------------------------------------------------------------------------|
  | si bulkToSource = false  | bulk             | bulk                                 | 0     | sud   |
  |                          |                  |                                      | 1     | sud   |
  |                          |                  |                        ---------------------------------|
  |                          |                  |                        | si ring = true | 2     | nord  |
  |                          |                  |                        |                | 3     | nord  |
  |                          |                  |                        |                | 4     | ouest |
  |                          |                  |                        |                | 5     | est   |
  ------------------------------------------------------------------------------------------------------


              source       gate1     gate2      bulk      bulk      drain1      drain2
             (source.1)   (gate.1)  (gate2.1)  (bulk.2)  (bulk.3)  (drain1.1)  (drain2.1)
                 |           |         |          |         |          |           |
              ___|___________|_________|__________|_________|__________|___________|___
              |  |           |         |                               |           |  |
              |  |           |         |                               |           |  |
              |  |           :---------------------G1------------------ |           |  |
              |  |           |         | ---------D2-------------------------------:  |
              |  :-------------------------------S------------------- |           |  |
              |  |           |         |                               |           |  |
              |  |           |         |                               |           |  |
    (bulk.4) --|  |           |         |                               |           |  |-- (bulk.5)
              |  |           |         |                               |           |  |
              |  :-------------------------------S------------------- |           |  |
              |  |           |         | ---------D1------------------:           |  |
              |  |           |         :----------G2------------------ |           |  |
              |  |           |         |                               |           |  |
              |__|___________|_________|_______________________________|___________|__|
                 |           |         |          |         |          |           |
              source       gate1     gate2      bulk      bulk      drain1      drain2
             (source.0)   (gate.0)  (gate2.0)  (bulk.0)  (bulk.1)  (drain1.0)  (drain2.0)


Parametres
----------
```

	Parametre	Type	Classe	Type C++	Unite	Valeur nominale	Statut possible
Entrees	TEMP	CP_IN	CP_SECONDARY	double	Kelvin	300.15 K	-
	VDS	CP_IN	CP_SECONDARY	double	Volt	techno	-
	VBS	CP_IN	CP_SECONDARY	double	Volt	techno	-
	VEG	CP_IN	CP_SECONDARY	double	Volt	techno	-
	NFINGER	CP_IN	CP_SECONDARY	long	-	-	-
Sorties	VTH	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID CP_NAV
	VDSAT	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID CP_NAV
	CGD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CGS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CGB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CBD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CSD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CM	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CMB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CMX	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	CBS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID CP_NAV
	MODE	CP_OUT	CP_SECONDARY	string	---	techno	CP_NAV "cut off" "subthreshold" "strong inversion & linear" "strong inversion & saturation" "weak inversion & linear" "weak inversion & saturation" "Unknown mode"
Entrees-sorties	W	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV CP_VALID CP_TOO_SMALL
	L	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV CP_VALID CP_TOO_SMALL
Mutable	IBIAS	CP_INOUT	CP_SECONDARY	double	Ampere	techno	CP_VALID CP_TOO_LARGE
		CP_OUT	CP_SECONDARY	double	Ampere	techno	CP_VALID CP_TOO_LARGE
	VGS	CP_IN	CP_SECONDARY	double	Volt	techno	-
		CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID CP_NAV
	GDS	CP_IN	CP_SECONDARY	double	Siemens	techno	-
		CP_OUT	CP_SECONDARY	double	Siemens	techno	CP_VALID CP_NAV
	GM	CP_IN	CP_SECONDARY	double	Siemens	techno	-
		CP_OUT	CP_SECONDARY	double	Siemens	techno	CP_VALID CP_NAV

VDD et VSS en variables globales CAIRO+

Contraintes

Parametre	Contraintes
L	L >= Lmin (contrainte interne)
W	W >= Wmin * 2.0 (contrainte interne)

DSES

Parametres fournis en entree	Parametres calcules	Nom de procedure	Classe	Etat possibles
:	VTH VDSAT IBIAS			
:				
:	GM GDS			

			CGS CGB CGD CBD CSD CM			
		VGS W L	CMB CMX CBS	IBIAS(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			MODE			
			L			CP_OK
			VTH VDSAT IBIAS			1
						2
			GM GDS			3
Groupe 1 VGS	TEMP VDS VBS	VGS W IBIAS	CGS CGB CGD CBD CSD CM	L(VGS,W,IBIAS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			W			CP_OK
			VTH VDSAT IBIAS			4
						5
			GM GDS			6
		VGS L IBIAS	CGS CGB CGD CBD CSD CM	W(VGS,L,IBIAS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			VTH	VTH(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			VTH VDSAT	VDSAT(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			VTH VDSAT MODE	MODE(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			GM	GM(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			GDS	GDS(VGS,W,L)	CP_UNCOMPLETE	CP_OK
		VGS W L	CGS CGB CGD CBD CSD CM	CAPA(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			CMB CMX CBS			
			VTH VDSAT IBIAS			
			GM GDS			
			CGS CGB CGD CSD CM			
			CMB CMX CBS	ALL(VGS,W,L)	CP_UNCOMPLETE	CP_OK
			MODE			
			VDSAT IBIAS			
			VTH VGS			
			GM GDS			
		VEG W L	CGS CGB CGD CBD CSD CM	IBIAS(VEG,W,L)	CP_UNCOMPLETE	CP_OK
			CMB CMX CBS			
			MODE			
			L			CP_OK
			VDSAT VTH VGS IBIAS			7
						8
Groupe 2 VEG	TEMP VDS VBS		GM GDS			9
		VEG W IBIAS	CGS CGB CGD CBD CSD CM	L(VEG,W,IBIAS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			W			CP_OK
			VDSAT VTH VGS IBIAS			10
						11
			GM GDS			12
		VEG L IBIAS	CGS CGB CGD CBD CSD CM	W(VEG,L,IBIAS)	CP_UNCOMPLETE	
			CMB CMX CBS			
			MODE			
			L			CP_OK
			VTH VDSAT IBIAS			1
						2
			GM			3

		:				
		: VGS GDS W	CGS CGB CGD CBD CSD CM	L(VGS,W,GDS)	CP_UNCOMPLETE	
		:	CMB CMX CBS			
		:				
		:	MODE			
Groupe 3	TEMP VDS VBS	:---	---	---	---	---
		:	W			
		:				CP_Ok
		:	VTH VDSAT IBIAS			4
		:				5
		:	GDS			6
		:				
		: VGS GM L	CGS CGB CGD CBD CSD CM	W(VGS,L,GM)	CP_UNCOMPLETE	
		:	CMB CMX CBS			
		:				
		:	MODE			
Directe	W	L	W L [NFINGER]	direct_size	CP_COMPLETE	13 14

Note: NFINGER est un parametre optionnel permettant de fixer le nombre de repliements

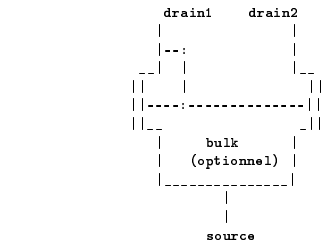
Messages

Etat de retour	Message
1	"L(VGS,W,IBIAS) : IBIAS is too large, try to decrease current"
2	"L(VGS,W,IBIAS) : IBIAS is too small, try to increase current"
3	"L(VGS,W,IBIAS) : L is too small for the current, try to increase IBIAS"
4	"W(VGS,L,IBIAS) : IBIAS is too large, try to decrease current"
5	"W(VGS,L,IBIAS) : IBIAS is too small, try to increase current"
6	"W(VGS,L,IBIAS) : W is too small for the current, try to increase IBIAS"
7	"L(VEG,W,IBIAS) : IBIAS is too large, try to decrease current"
8	"L(VEG,W,IBIAS) : IBIAS is too small, try to increase current"
9	"L(VEG,W,IBIAS) : L is too small for the current, try to increase IBIAS"
10	"W(VEG,L,IBIAS) : IBIAS is too large, try to decrease current"
11	"W(VEG,L,IBIAS) : IBIAS is too small, try to increase current"
12	"W(VEG,L,IBIAS) : W is too small for the current, try to increase IBIAS"
13	"direct_size : W is too small - DRM rules"
14	"direct_size : L is too small - DRM rules"

Description: Miroir de courant simple

Generateur CM_MOS

Schema electrique



Creation

CAIRO_CREATE(char *libName,char *genName,char *name,char *trName1,char *trName2,char type,bool bulkToSource)

libName = "libMOS"
genName = "CM_MOS"
name = nom d'instance
trName1 = nom du transistor 1
trName2 = nom du transistor 2
type = TRANSN ou TRANSP
bulkToSource = true ou false (source reliee en interne au bulk si bulkToSource = true. Dans ce cas pas de connecteur bulk.)

Ordre des connecteurs pour l'instanciation:

```
  Si bulkToSource = false   :    1 drain1
                                 2 drain2
                                 3 source
                                 4 bulk

  Si bulkToSource = true    :    1 drain1
                                 2 drain2
                                 3 source
```

Modes de realisation
---

* Parametres

```
  ring         = true ou false (presence d'un anneau de garde si ring = true)
  dummy        = true ou false (presence de transistors fictifs sur les bords si dummy = true)
  stack        = -1 ou > 1 (nombre d'empilements. -1 vaut pour un empilement libre, sinon vaut
                 le nombre d'empilements desire)
```

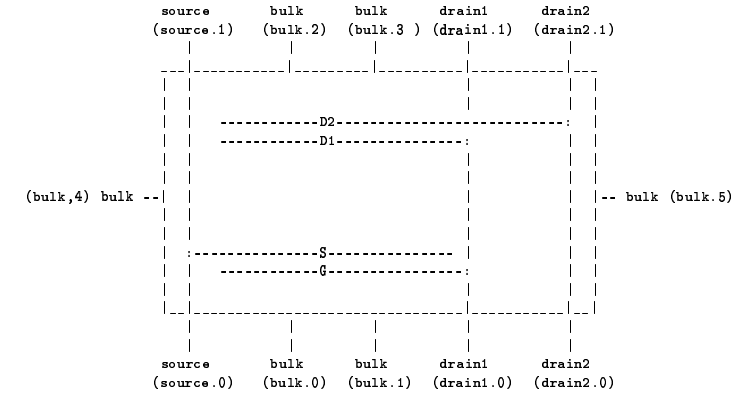
* Arborescence:

```
BEGIN_CIRCUIT nom_circuit
  INSTANCE nom_instance (dummy,false) (ring,false) (stack,-1)
END_CIRCUIT
```

Plan de masse

Les connecteurs physiques sont au nord et au sud (acces par les index)

Nom des connecteurs electriques	Nom des connecteurs physiques correspondant	Index	Face
source	source	0	sud
		1	nord
drain1	drain1	0	sud
		1	nord
drain2	drain2	0	sud
		1	nord
si bulkToSource = false	bulk	bulk	0	sud
			1	sud
		si ring = true	2	nord
			3	nord
			4	ouest
			5	est



Parametres

	Parametre	Type	Classe	Type C++	Unite	Valeur nominale	Statut possible
Entrees	TEMP	CP_IN	CP_SECONDARY	double	Kelvin	300.15 K	-
	VDS	CP_IN	CP_SECONDARY	double	Volt	techno	-

	NFINGER	CP_IN	CP_SECONDARY	long	-	-	-
	VTH	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID
							CP_NAV
	VDSAT	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID
							CP_NAV
	CGD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CGS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CGB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
Sorties	CBD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CSD	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CM	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CMB	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CMX	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	CBS	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
	MODE	CP_OUT	CP_SECONDARY	string	---	techno	"cut off"
							"subthreshold"
							"strong inversion & linear"
							"strong inversion & saturation"
							"weak inversion & linear"
							"weak inVersion & saturation"
							"Unknown mode"
	W	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV
							CP_VALID
Entrees-sorties							CP_TOO_SMALL
	L	CP_INOUT	CP_PRIMARY	double	Metre	techno	CP_NAV
							CP_VALID
							CP_TOO_SMALL
	IREF	CP_INOUT	CP_SECONDARY	double	Ampere	techno	CP_VALID
							CP_TOO_LARGE
		CP_OUT	CP_SECONDARY	double	Ampere	techno	CP_VALID
Mutable							CP_TOO_LARGE
	GM	CP_IN	CP_SECONDARY	double	Siemens	techno	-
		CP_OUT	CP_SECONDARY	double	Siemens	techno	CP_VALID
							CP_NAV

VDD et VSS en variables globales CAIRO+

Contraintes

Parametre	Contraintes
L	L >= Lmin (contrainte interne)
W	W >= Wmin * 2.0 (contrainte interne)

DSES

	Parametres fournis en entree		Parametres calcules	Nom de procedure	Classe	Etat possibles
		:	VTH VDSAT IREF			
		:				
		:	GM GDS			
		:				
		:	CGS CGB CGD CBD CSD CM			
		: W L	CMB CMX CBS	IREF(W,L)	CP_UNCOMPLETE	CP_OK
		:				
		:	MODE			
		:	L			
		:				CP_OK
		:	VTH VDSAT IREF			1
		:				2
		:	GM GDS			3
Groupe 1		:				
	TEMP VDS	: W IREF	CGS CGB CGD CBD CSD CM	L(W,IREF)	CP_UNCOMPLETE	
		:	CMB CMX CBS			

		:					
		:	MODE				
		:---					
		:	W				
		:				CP_OK	
		:	VTH VDSAT IREF			4	
		:				5	
		:	GM GDS			6	
		:					
		: L IREF	CGS CGB CGD CBD CSD CM	W(L,IREF)	CP_UNCOMPLETE		
		:	CMB CMX CBS				
		:					
		:	MODE				
		:---					
		:	VTH	VTH(W,L)	CP_UNCOMPLETE	CP_OK	
		:	---				
		:	VTH VDSAT	VDSAT(W,L)	CP_UNCOMPLETE	CP_OK	
		:	---				
		:	VTH VDSAT MODE	MODE(W,L)	CP_UNCOMPLETE	CP_OK	
		:	---				
		:	GM	GM(W,L)	CP_UNCOMPLETE	CP_OK	
		:	---				
		:	GDS	GDS(W,L)	CP_UNCOMPLETE	CP_OK	
		:	---				
		: W L	CGS CGB CGD CBD CSD CM	CAPA(W,L)	CP_UNCOMPLETE	CP_OK	
		:	CMB CMX CBS				
		:	---				
		:	VTH VDSAT IREF				
		:					
		:	GM GDS				
		:					
		:	CGS CGB CGD CBD CSD CM				
		:	CMB CMX CBS	ALL(W,L)	CP_UNCOMPLETE	CP_OK	
		:					
		:	MODE				
		:===					
		:	L				
		:					
		:	VTH VDSAT IREF				
		:					
		:	GM				
		:					
		: GDS W	CGS CGB CGD CBD CSD CM	L(W,GDS)	CP_UNCOMPLETE		
		:	CMB CMX CBS				
		:					
		:	MODE				
Groupe 2	TEMP VDS	:---					
		:	W				
		:					
		:	VTH VDSAT IREF				
		:					
		:	GDS				
		:					
		: GM L	CGS CGB CGD CBD CSD CM	W(L,GM)	CP_UNCOMPLETE		
		:	CMB CMX CBS				
		:					
		:	MODE				
		:===					
Directe	W L		W L [NFINGER]	direct_size	CP_COMPLETE	7 8	

Note: NFINGER est un parametre optionnel permettant de fixer le nombre de repliements

Messages

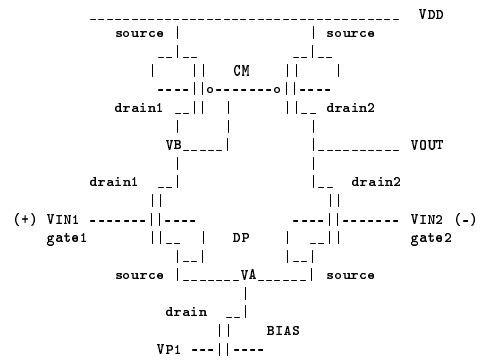
Etat de retour	Message
1	"L(W,IREF) : IREF is too large, try to decrease current"
2	"L(W,IREF) : IREF is too small, try to increase current"
3	"L(W,IREF) : L is too small for the current, try to increase IREF"
4	"W(L,IREF) : IREF is too large, try to decrease current"
5	"W(L,IREF) : IREF is too small, try to increase current"
6	"W(L,IREF) : W is too small for the current, try to increase IREF"
7	"direct_size : W is too small - DRM rules"
8	"direct_size : L is too small - DRM rules"

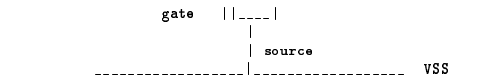
Annexe C

Mode d'emploi de l'OTA

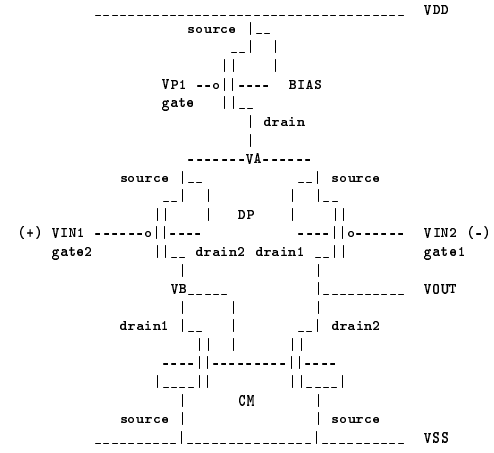
```
-------------------------------------------------------------------------------
| Toute copie, reproduction, meme partielle est interdite sans le consentement  |
| express de l'auteur. Toute modification lors de la reproduction est interdite |
-------------------------------------------------------------------------------

Redacteur        Pierre Nguyen Tuong
Version initiale 1.0  12/01/2005 CAIRO+ version 8.2.0  Okolona
Revision         1.1  09/02/2005 CAIRO+ version 9.0.0  Guard Hill
Revision         1.2  06/04/2005 CAIRO+ version 9.0.0  Guard Hill
Revision         2.0  03/08/2005 CAIRO+ version 10.0.0 Honey Springs
Revision         2.1  17/10/2005 CAIRO+ version 10.0.0 Honey Springs
Revision         2.1  17/10/2005 CAIRO+ version 11.0.1 LaFourche Crossing

                         ---------------------------------
                         | Generateurs          | Version |
-------------------------------------------------|---------|
| Bibliotheque  | libMOS  |                     |         |
|-------------------------| OTA                 |         |
| Technologie   | MOS     |                     |         |
|=================================================| CAIRO+  |
| Inventeurs    | License | Copyright           |         |
|-------------------------------------------------|         |
| PNT           |         |                     |
|               | QPL     | ASIM                |         |
| Conseils:RI   |         |                     |         |
|          MML  |         |                     |         |
-------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------------------
Description: amplificateur operationnel a transconductance non differentiel
-------------------------------------------------------------------------------------------------------------
Generateur OTA
-------------------------------------------------------------------------------------------------------------

Schema electrique
-----------------

OTA de type N:

                        ------------------------------------  VDD
                           source |                | source
                                __|__           __|__
                               |    ||   CM    ||    |
                                ----||o-------o||----
                           drain1 __||  |      ||__ drain2
                                  |     |         |
                                 VB_____|         |__________ VOUT
                                  |               |
                        drain1  __|               |__  drain2
                               ||                   ||
                (+) VIN1 -------||----           ----||------- VIN2 (-)
                    gate1       ||__  |   DP    |  __||        gate2
                                  |__|         |__|
                           source |_______VA______| source
                                         |
                                 drain  __|
                                       ||    BIAS
                                VP1 ---||----
```





OTA de type P:



Creation

CAIRO_CREATE(char *libName,char *genName,char *name,char type)

libName = "libOTA"
genName = "OTA"
name = nom d'instance
type = TRANSN ou TRANSP

Ordre des connecteurs pour l'instanciation:

vp1
vin1
vin2
vout
vdd
vss

Modes de realisation
----

* Parametres

ring = true ou false (presence d'un anneau de garde si ring = true)
dummy = true ou false (presence de transistors fictifs sur les bords si dummy = true)
stack = -1 ou > 1 (nombre d'empilements. -1 vaut pour un empilement libre, sinon vaut le nombre d'empilements desire)

* Arborescence:

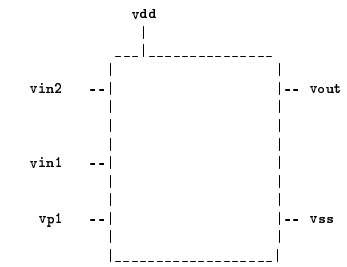
```
BEGIN_CIRCUIT nom_circuit
INSTANCE OTA_BIAS (dummy,false) (ring,false)
INSTANCE OTA_DP (dummy,false) (ring,false) (stack,-1)
INSTANCE OTA_CM (dummy,false) (ring,false) (stack,-1)
END_CIRCUIT
```

Plan de masse

OTA de type N:

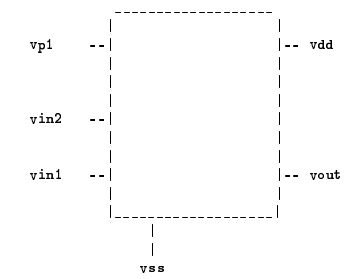
Nom des connecteurs electriques	Nom des connecteurs physiques correspondant	Index	Face
vp1	vp1	0	ouest
vin1	vin1	0	ouest
vin2	vin2	0	ouest

vout	vout	0	est
vdd	vdd	0	nord
vss	vss	0	est



OTA de type P:

Nom des connecteurs electriques	Nom des connecteurs physiques correspondant	Index	Face
vp1	vp1	0	ouest
vin1	vin1	0	ouest
vin2	vin2	0	ouest
vout	vout	0	est
vdd	vdd	0	est
vss	vss	0	sud



Parametres

	Parametre	Type	Classe	Type C++	Unite	Valeur nominale	Statut possible
Entrees	TEMP	CP_IN	CP_SECONDARY	double	Kelvin	300.15 K	-
	VEG_BIAS	CP_IN	CP_SECONDARY	double	Volt	techno	-
	VEG_DP	CP_IN	CP_SECONDARY	double	Volt	techno	-
	L_INIT (1)	CP_IN	CP_SECONDARY	double	Metre	techno	-
	VIN	CP_IN	CP_SECONDARY	double	Volt	techno	-
	VOUT	CP_IN	CP_SECONDARY	double	Volt	techno	-
	CLOAD	CP_IN	CP_SECONDARY	double	Farad	techno	-
Sorties	VP1	CP_OUT	CP_SECONDARY	double	Volt	techno	CP_VALID CP_NAV
	UGF	CP_OUT	CP_PRIMARY	double	Hz	techno	CP_VALID CP_NAV CP_TOO_LARGE CP_TOO_SMALL
	PM	CP_OUT	CP_PRIMARY	double	Degres	techno	CP_VALID CP_NAV CP_TOO_LARGE CP_TOO_SMALL
	GBW	CP_OUT	CP_SECONDARY	double	Hz	techno	CP_VALID CP_NAV
	COF	CP_OUT	CP_SECONDARY	double	Hz	techno	CP_VALID

							CP_NAV
	SR	CP_OUT	CP_SECONDARY	double	Volt/Sec	techno	CP_VALID
							CP_NAV
	GAIN	CP_OUT	CP_SECONDARY	double	db	techno	CP_VALID
							CP_NAV
	CLOAD_OUT	CP_OUT	CP_SECONDARY	double	Farad	techno	CP_VALID
							CP_NAV
Entrees-sorties	IBIAS	CP_INOUT	CP_SECONDARY	double	Ampere	techno	CP_VALID
							CP_NAV
							CP_TOO_LARGE
							CP_TOO_SMALL

```
TEMP      = temperature
IBIAS     = courant de polarisation (Biasing current)
VEG_BIAS  = VEG du transistor du device BIAS
VEG_DP    = VEG des transistors du device DP
L_INIT    = L initial
VIN       = tension de mode commun en entree
VOUT      = tension de mode commun en sortie
CLOAD     = capacite de charge
VP1       = tension de polarisation
UGF       = frequence de transistion (Unity Gain Frequency)
PM        = marge de phase (Phase Margin)
GBW       = produit gain-bande (Gain Bandwidth)
COF       = frequence de coupure (Cut-Off Frequency)
SR        = Vitesse maximale de variation de la tension de sortie (Slew-Rate)
GAIN      = gain statique
CLOAD_OUT = Somme des capacites parasites de routage et de CLOAD


(1) L_INIT est optionnel

VDD et VSS en variables globales CAIRO+

VIN et VOUT sont donnes en valeur absolue
```

Contraintes

Parametre	Contraintes
UGF	UGF >= UGF_min (obligatoire)
PM	PM_low <= PM (obligatoire)
	ou
	PM_low <= PM <= PM_high (obligatoire)
CLOAD_OUT	CLOAD_OUT <= CLOAD_max (obligatoire)
SR	SR >= SR_min (optionnel)
GAIN	GAIN >= GAIN_min (optionnel)

DSES

Parametres fournis en entree	Parametres calcules	Nom de procedure	Classe	Etats possibles
: VBS IBIAS				CP_OK
: VIN VOUT				1 a 9
: VEG_BIAS	VP1	VP1	CP_UNCOMPLETE	16 a 30
: VEG_DP				
: L_INIT (opt)				
: (VBS) (IBIAS)				CP_OK
TEMP : (VEG_BIAS)	UGF PM CLOAD_OUT IBIAS	UGF	CP_COMPLETE	10 a 30
: (VEG_DP)	SR GAIN COF GBW (VP1)			
: CLOAD				
: (VBS) (IBIAS)				CP_OK
: (VEG_BIAS)	UGF PM CLOAD_OUT IBIAS	UGF_MIN_PM	CP_COMPLETE	10 a 30
: (VEG_DP)	SR GAIN COF GBW (VP1)			
: CLOAD				

```
Note: UGF et UGF_MIN_PM recalculent VP1. Il n'est pas necessaire d'appeler VP1 avant UGF (neanmoins, les
      parametres de VP1 doivent avoir ete positionnes avant d'appeler UGF ou UGF_MIN_PM)
    : UGF_MIN_PM minimise la marge de phase
```

Messages

Etat de retour	Message
1	"VP1: DP: IBIAS is too large, try to decrease current"
2	"VP1: DP: IBIAS is too small, try to increase current"
3	"VP1: DP: W of DP is too small, try to increase current"
4	"VP1: BIAS: IBIAS is too large, try to decrease current"
5	"VP1: BIAS: IBIAS is too small, try to increase current"
6	"VP1: BIAS: W of DP is too small, try to increase current"
7	"VP1: CM: IBIAS is too large, try to decrease current"
8	"VP1: CM: IBIAS is too small, try to increase current"
9	"VP1: CM: W of CM is too small, try to increase current"
10	"UGF: IBIAS is too large, try to decrease current or UGF or increase PM"
11	"UGF: IBIAS is too small, try to increase current or UGF or decrease PM"
12	"UGF: IBIAS is too large in PM loop (L too small), try to decrease current or UGF or increase PM"
13	"UGF: IBIAS too small, SR too small, SR unsatisfied"
14	"UGF: IBIAS too small, GAIN too small, GAIN unsatisfied"
15	"VP1: BIAS: transistor mode is cut off, decrease VEG (VDS should be over VEG)"
16	"VP1: BIAS: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"
17	"VP1: BIAS: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"
18	"VP1: BIAS: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"
19	"VP1: BIAS: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)"
20	"VP1: DP: transistor mode is cut off, decrease VEG (VDS should be over VEG)"
21	"VP1: DP: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"
22	"VP1: DP: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"
23	"VP1: DP: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"
24	"VP1: DP: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)"
25	"VP1: CM: transistor mode is cut off, decrease VEG (VDS should be over VEG)"
26	"VP1: CM: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"
27	"VP1: CM: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"
28	"VP1: CM: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"
29	"VP1: CM: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)"

Annexe D

Code source du générateur OTA

```
/*************************************************************************/
/*                        Fichier : ota.cpp                              */
/*                                                                       */
/* Description : OTA simple non differentiel                             */
/*                                                                       */
/* Langage : C++                Version : 2.1             Date : 17/10/2005 */
/* Auteur  : Pierre Nguyen Tuong               Qualif: Pierre Nguyen Tuong */
/*                                                                       */
/* Conseils: Ramy Iskander                                               */
/*           Marie-Minerve Louerat                                       */
/*                                                                       */
/* License : QPL                                                         */
/*                                                                       */
/* Historique                                                            */
/* 1.0   - 28/12/2004 : version initiale                                 */
/* 2.0   - 06/04/2005 : version plus juste et amelioree                  */
/* 2.1   - 17/10/2005 : version encore plus juste et amelioree           */
/*                                                                       */
/* Fonctions :                                                           */
/*                                                                       */
/*************************************************************************/

#include "cairoplus.h"

// ------------------------------------------------------------------------------------------------------------------
// Description: amplificateur operationnel a transconductance non differentiel
// ------------------------------------------------------------------------------------------------------------------
// Generateur OTA
// ------------------------------------------------------------------------------------------------------------------
//
// Schema electrique
// -----------------
//
// OTA de type N:
//
//                          ------------------------------------  VDD
//                             source |               | source
//                                  __|__           __|__
//                                 |    ||   CM    ||    |
//                                  ----||o-------o||----
//                             drain1 __||  |      ||__ drain2
//                                    |     |         |
//                                   VB_____|         |__________ VOUT
//                                    |               |
//                          drain1  __|               |__  drain2
//                                    ||                ||
//                 (+) VIN1 -------||----            ----||------- VIN2 (-)
//                     gate1       ||__  |   DP    |  __||        gate2
//                                   |__|         |__|
//                                 source |_______VA______|  source
//                                                 |
//                                       drain  __|
//                                              ||    BIAS
//                                     VP1 ---||----
//                                     gate   ||____|
//                                              |
//                                              | source
//                          ------------------|------------------  VSS
//
//
//  OTA de type P:
```

```
//
//                        ------------------------------------  VDD
//                                  source |__
//                                       __|  |
//                                      ||    |
//                                VP1 --o||----  BIAS
//                                gate   ||__
//                                           | drain
//                                           |
//                                     -------VA------
//                                source |__           __| source
//                                     __|  |         |  |__
//                                    ||    |   DP    |    ||
//                (+) VIN1 ------o||----           ----||o------  VIN2 (-)
//                    gate2       ||__ drain2 drain1 __||         gate1
//                                      |               |
//                                       VB_____        |__________  VOUT
//                                        |     |       |
//                                drain1 |__   |       __| drain2
//                                         ||  |      ||
//                                     ----||---------||----
//                                    |____||         ||____|
//                                         |      CM       |
//                                source |               | source
//                                ----------|---------------|----------  VSS
//
//
// Creation
// --------
//
//   CAIRO_CREATE(char *libName,char *genName,char *name,char type)
//
//   libName      = "libOTA"
//   genName      = "OTA"
//   name         = nom d'instance
//   type         = TRANSN ou TRANSP
//
//   Ordre des connecteurs pour l'instanciation:
//
//     vp1
//     vin1
//     vin2
//     vout
//     vdd
//     vss
//
//
// Modes de realisation
// --------------------
//
// * Parametres
//
//   ring         = true ou false (presence d'un anneau de garde si ring = true)
//   dummy        = true ou false (presence de transistors fictifs sur les bords si dummy = true)
//   stack        = -1 ou > 1 (nombre d'empilements. -1 vaut pour un empilement libre, sinon vaut
//                  le nombre d'empilements desire)
//
// * Arborescence:
//
// BEGIN_CIRCUIT nom_circuit
// INSTANCE OTA_BIAS (dummy,false) (ring,false)
// INSTANCE OTA_DP (dummy,false) (ring,false) (stack,-1)
// INSTANCE OTA_CM (dummy,false) (ring,false) (stack,-1)
// END_CIRCUIT
//
//
// Plan de masse
// -------------
//
// OTA de type N:
//
//      ------------------------------------------------------------------------------
//      | Nom des connecteurs | Nom des connecteurs                     | Index | Face  |
//      |     electriques     | physiques correspondant                 |       |       |
//      |----------------------------------------------------------------------------|
//      | vp1                 | vp1                                     | 0     | ouest |
//      |----------------------------------------------------------------------------|
//      | vin1                | vin1                                    | 0     | ouest |
//      |----------------------------------------------------------------------------|
//      | vin2                | vin2                                    | 0     | ouest |
//      |----------------------------------------------------------------------------|
//      | vout                | vout                                    | 0     | est   |
//      |----------------------------------------------------------------------------|
//      | vdd                 | vdd                                     | 0     | nord  |
//      |----------------------------------------------------------------------------|
//      | vss                 | vss                                     | 0     | est   |
//      ------------------------------------------------------------------------------
//
```

```
//
//                              vdd
//                               |
//                            ___|_______________
//                           |                   |
//                  vin2   --|                   |-- vout
//                           |                   |
//                           |                   |
//                           |                   |
//                  vin1   --|                   |
//                           |                   |
//                           |                   |
//                   vp1   --|                   |-- vss
//                           |                   |
//                           |___________________|
//
//
// OTA de type P:
//
//      ------------------------------------------------------------------------------
//      | Nom des connecteurs | Nom des connecteurs                    | Index | Face  |
//      |     electriques     | physiques correspondant                |       |       |
//      |-----------------------------------------------------------------------------|
//      | vp1                 | vp1                                    | 0     | ouest |
//      |-----------------------------------------------------------------------------|
//      | vin1                | vin1                                   | 0     | ouest |
//      |-----------------------------------------------------------------------------|
//      | vin2                | vin2                                   | 0     | ouest |
//      |-----------------------------------------------------------------------------|
//      | vout                | vout                                   | 0     | est   |
//      |-----------------------------------------------------------------------------|
//      | vdd                 | vdd                                    | 0     | est   |
//      |-----------------------------------------------------------------------------|
//      | vss                 | vss                                    | 0     | sud   |
//      ------------------------------------------------------------------------------
//
//
//                           ___________________
//                          |                   |
//                  vp1   --|                   |-- vdd
//                          |                   |
//                          |                   |
//                          |                   |
//                  vin2  --|                   |
//                          |                   |
//                          |                   |
//                  vin1  --|                   |-- vout
//                          |                   |
//                          |___________________|
//                              |
//                              |
//                             vss
//
// Parametres
// ----------
//
//                     ------------------------------------------------------------------------------------------------------
//                     | Parametre | Type     | Classe        | Type C++ | Unite   | Valeur   | Statut possible                 |
//                     |           |          |               |          |         | nominale |                                 |
//   ------------------|-----------|----------|---------------|----------|---------|----------|---------------------------------|
//   |                 | TEMP      | CP_IN    | CP_SECONDARY  | double   | Kelvin  | 300.15 K | -                               |
//   |                 | VEG_BIAS  | CP_IN    | CP_SECONDARY  | double   | Volt    | techno   | -                               |
//   | Entrees         | VEG_DP    | CP_IN    | CP_SECONDARY  | double   | Volt    | techno   | -                               |
//   |                 | L_INIT (1)| CP_IN    | CP_SECONDARY  | double   | Metre   | techno   | -                               |
//   |                 | VIN       | CP_IN    | CP_SECONDARY  | double   | Volt    | techno   | -                               |
//   |                 | VOUT      | CP_IN    | CP_SECONDARY  | double   | Volt    | techno   | -                               |
//   |                 | CLOAD     | CP_IN    | CP_SECONDARY  | double   | Farad   | techno   | -                               |
//   |-----------------|-----------|----------|---------------|----------|---------|----------|---------------------------------|
//   |                 | VP1       | CP_OUT   | CP_SECONDARY  | double   | Volt    | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 | UGF       | CP_OUT   | CP_PRIMARY    | double   | Hz      | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 |           |          |               |          |         |          | CP_TOO_LARGE                    |
//   |                 |           |          |               |          |         |          | CP_TOO_SMALL                    |
//   | Sorties         | PM        | CP_OUT   | CP_PRIMARY    | double   | Degres  | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 |           |          |               |          |         |          | CP_TOO_LARGE                    |
//   |                 |           |          |               |          |         |          | CP_TOO_SMALL                    |
//   |                 | GBW       | CP_OUT   | CP_SECONDARY  | double   | Hz      | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 | COF       | CP_OUT   | CP_SECONDARY  | double   | Hz      | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 | SR        | CP_OUT   | CP_SECONDARY  | double   | Volt/Sec| techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 | GAIN      | CP_OUT   | CP_SECONDARY  | double   | db      | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
//   |                 | CLOAD_OUT | CP_OUT   | CP_SECONDARY  | double   | Farad   | techno   | CP_VALID                        |
//   |                 |           |          |               |          |         |          | CP_NAV                          |
```

```
//   |-------------------------------------------------------------------------------------------------------------------|
//   | Entrees-sorties | IBIAS     | CP_INOUT | CP_SECONDARY | double   | Ampere  | techno   | CP_VALID                  |
//   |                 |           |          |              |          |         |          | CP_NAV                    |
//   |                 |           |          |              |          |         |          | CP_TOO_LARGE              |
//   |                 |           |          |              |          |         |          | CP_TOO_SMALL              |
//   -------------------------------------------------------------------------------------------------------------------
//
//   TEMP      = temperature
//   IBIAS     = courant de polarisation (Biasing current)
//   VEG_BIAS  = VEG du transistor du device BIAS
//   VEG_DP    = VEG des transistors du device DP
//   L_INIT    = L initial
//   VIN       = tension de mode commun en entree
//   VOUT      = tension de mode commun en sortie
//   CLOAD     = capacite de charge
//   VP1       = tension de polarisation
//   UGF       = frequence de transition (Unity Gain Frequency)
//   PM        = marge de phase (Phase Margin)
//   GBW       = produit gain-bande (Gain Bandwidth)
//   COF       = frequence de coupure (Cut-Off Frequency)
//   SR        = Vitesse maximale de variation de la tension de sortie (Slew-Rate)
//   GAIN      = gain statique
//   CLOAD_OUT = Somme des capacites parasites de routage et de CLOAD
//
//
//   (1) L_INIT est optionnel
//
//   VDD et VSS en variables globales CAIRO+
//
//   VIN et VOUT sont donnes en valeur absolue
//
//
// Contraintes
// -----------
//
//   ---------------------------------------------------------------------
//   |  Parametre  | Contraintes                                         |
//   |-------------------------------------------------------------------|
//   | UGF         | UGF >= UGF_min (obligatoire)                        |
//   |-------------------------------------------------------------------|
//   | PM          | PM_low <= PM (obligatoire)                          |
//   |             |             ou                                      |
//   |             | PM_low <= PM <= PM_high (obligatoire)               |
//   |-------------------------------------------------------------------|
//   | CLOAD_OUT   | CLOAD_OUT <= CLOAD_max (obligatoire)                |
//   |-------------------------------------------------------------------|
//   | SR          | SR    >= SR_min (optionnel)                         |
//   |-------------------------------------------------------------------|
//   | GAIN        | GAIN  >= GAIN_min (optionnel)                       |
//   ---------------------------------------------------------------------
//
//
// DSES
// ----
//
//   ---------------------------------------------------------------------------------------------------------------------
//   |   Parametres fournis en entree   |   Parametres calcules      | Nom de procedure          | Classe         | Etats       |
//   |                                  |                            |                           |                | possibles   |
//   |-------------------------------------------------------------------------------------------------------------------|
//   |                  : VBS IBIAS     |                            |                           |                | CP_OK       |
//   |                  : VIN VOUT      |                            |                           |                | 1 a 9       |
//   |                  : VEG_BIAS      |   VP1                      | VP1                       | CP_UNCOMPLETE  | 16 a 30     |
//   |                  : VEG_DP        |                            |                           |                |             |
//   |                  : L_INIT (opt)  |                            |                           |                |             |
//   |                  :---------------------------------------------------------------------------------------------|
//   |                  : (VBS) (IBIAS) |                            |                           |                | CP_OK       |
//   |  TEMP            : (VEG_BIAS)    |   UGF PM CLOAD_OUT IBIAS   | UGF                       | CP_COMPLETE    | 10 a 30     |
//   |                  : (VEG_DP)      |   SR GAIN COF GBW (VP1)    |                           |                |             |
//   |                  : CLOAD         |                            |                           |                |             |
//   |                  :---------------------------------------------------------------------------------------------|
//   |                  : (VBS) (IBIAS) |                            |                           |                | CP_OK       |
//   |                  : (VEG_BIAS)    |   UGF PM CLOAD_OUT IBIAS   | UGF_MIN_PM                | CP_COMPLETE    | 10 a 30     |
//   |                  : (VEG_DP)      |   SR GAIN COF GBW (VP1)    |                           |                |             |
//   |                  : CLOAD         |                            |                           |                |             |
//   ---------------------------------------------------------------------------------------------------------------------
//
//   Note: UGF et UGF_MIN_PM recalculent VP1. Il n'est pas necessaire d'appeler VP1 avant UGF (neanmoins, les
//         parametres de VP1 doivent avoir ete positionnes avant d'appeler UGF ou UGF_MIN_PM)
//       : UGF_MIN_PM minimise la marge de phase
//
//
// Messages
// --------
//
//   ---------------------------------------------------------------------------------------------------------
//   | Etat de retour | Message                                                                              |
//   |-------------------------------------------------------------------------------------------------------|
//   | 1              | "VP1: DP: IBIAS is too large, try to decrease current"                               |
```

```
//   | 2              | "VP1: DP: IBIAS is too small, try to increase current"                                          |
//   | 3              | "VP1: DP: W of DP is too small, try to increase current"                                        |
//   | 4              | "VP1: BIAS: IBIAS is too large, try to decrease current"                                        |
//   | 5              | "VP1: BIAS: IBIAS is too small, try to increase current"                                        |
//   | 6              | "VP1: BIAS: W of DP is too small, try to increase current"                                      |
//   | 7              | "VP1: CM: IBIAS is too large, try to decrease current"                                          |
//   | 8              | "VP1: CM: IBIAS is too small, try to increase current"                                          |
//   | 9              | "VP1: CM: W of CM is too small, try to increase current"                                        |
//   | 10             | "UGF: IBIAS is too large, try to decrease current or UGF or increase PM"                        |
//   | 11             | "UGF: IBIAS is too small, try to increase current or UGF or decrease PM"                        |
//   | 12             | "UGF: IBIAS is too large in PM loop (L too small), try to decrease current or UGF or increase PM"  |
//   | 13             | "UGF: IBIAS too small, SR too small, SR unsatisfied"                                            |
//   | 14             | "UGF: IBIAS too small, GAIN too small, GAIN unsatisfied"                                        |
//   | 15             | "VP1: BIAS: transistor mode is cut off, decrease VEG (VDS should be over VEG)"                  |
//   | 16             | "VP1: BIAS: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"             |
//   | 17             | "VP1: BIAS: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"  |
//   | 18             | "VP1: BIAS: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"    |
//   | 19             | "VP1: BIAS: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)" |
//   | 20             | "VP1: DP: transistor mode is cut off, decrease VEG (VDS should be over VEG)"                    |
//   | 21             | "VP1: DP: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"               |
//   | 22             | "VP1: DP: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"    |
//   | 23             | "VP1: DP: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"      |
//   | 24             | "VP1: DP: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)"  |
//   | 25             | "VP1: CM: transistor mode is cut off, decrease VEG (VDS should be over VEG)"                    |
//   | 26             | "VP1: CM: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)"               |
//   | 27             | "VP1: CM: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)"    |
//   | 28             | "VP1: CM: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)"      |
//   | 29             | "VP1: CM: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)"  |
//   ---------------------------------------------------------------------------------------------------------------------


// ---------------------------------------------------------------------------------------------------------
// ---------------------------------------------- Fonctions utiles ----------------------------------------
// ---------------------------------------------------------------------------------------------------------

// Calcul de w0 (zero) en rad/s
//           w1 (premier pole non dominant) en rad/s
//           w2 (second pole non dominant) en rad/s - frequence de coupure
//           G1 (conductance) en siemens
//           G2 (conductance) en siemens

static void fw0w1w2G1G2C2(double gds_dp,double cgd_dp,double cbd_dp,
                          double gds_cm,double gm_cm,double cgs_cm,double cgb_cm,double cbd_cm,double cgd_cm,
                          double cload,
                          double &w0,double &w1,double &w2,double &G1,double &G2,double &C2)
{
  double C1 = 0.0 ;

  // Elements intermediaires
  C1 = cgd_dp + cbd_dp + cgs_cm + cgb_cm + cbd_cm + cgs_cm + cgb_cm ;

  C2 = cgd_dp + cbd_dp + cgd_cm + cbd_cm + cload ;

  G1 = gds_dp + gds_cm + gm_cm ;

  G2 = gds_dp + gds_cm ;

  // Calcul des poles
  w1 = G1 / C1 ;

  w2 = (gds_dp + gds_cm) / C2 ;

  // Zero
  w0 = w1 * (1.0 + (gm_cm / G1)) ;
}

// Calcul de la pulsation au point de transition exprimee en rad/s (wt).
// La dichotomie sur l'equation exacte du gain est utilisee pour obtenir un wt plus precis.
//
// La pulsation au point de transition est obtenue lorsque |Ad| = 0 (le gain est nul). Comme Ad = f(w),
// il faut resoudre f(w) = 0. La solution est w = wt.
//
//        |                                    jw          |
//        |                               1 +  --          |
//        | 1  gm_dp . (gm_cm + G1)            w0          |
// |Ad| = | -  -------------------- . --------------------- |
//        | 2       G1 . G2           (     jw ) (     jw ) |
//        |                           ( 1 + -- ) ( 1 + -- ) |
//        |                           (     w1 ) (     w2 ) |
//
//                                                 ---------
//                                                /        2
//                                             |  /     (w )
//                                             | /  1 + (--)
//          1  gm_dp . (gm_cm + G1)            |/       (w0)
// |Ad| =   -  -------------------- . ------------------------------
//          2       G1 . G2                ---------        ---------
//                                        /        2       /        2
```

```
//                                  |  /     (w )    |  /     (w )
//                                  | /  1 + (--)  . | /  1 + (--)
//                                  |/       (w1)    |/       (w2)
//
//                              ---------
//                             /         2
//                        |  /     (w )
//                        | /  1 + (--)
//                        |/       (w0)                     gm_dp . (gm_cm + G1)
// |Ad| =   K . ------------------------------    avec K = --------------------
//                    ---------        ---------                   G1 . G2
//                  /         2      /         2
//              |  /     (w )    |  /     (w )
//              | /  1 + (--)  . | /  1 + (--)
//              |/       (w1)    |/       (w2)


// Gain dynamique en db

static double fgainWt(double K,double wt,double w0,double w1,double w2)
{
  double gain = 0.0 ; // Gain en db

  gain = 20.0 * log10(K * ( (sqrt(1 + pow(wt / w0,2.0))) / ( (sqrt(1 + pow(wt / w1,2.0))) * (sqrt(1 + pow(wt / w2,2.0))) ) )) ;

  return gain ;
}

// Pulsation au point de transition en rad/s

static double fwt(double gm_dp,double gm_cm,double G1,double G2,double w0,double w1,double w2,double C2)
{
  double       gain     = 0.0    ; // Gain en db
  double       K        = 0.0    ;
  double       wt       = 0.0    ; // Pulsation au point de transition en rad/s
  double       wt_left  = 0.0    ; // Borne gauche de l'intervalle
  double       wt_right = 0.0    ; // Borne droite de l'intervalle
  const double epsilon  = 0.0001 ; // Precision

  double       wt0_Hz   = 0.0    ;
  double       decade   = 1.0    ;

  // Facteur multiplicatif constant
  K = 0.5 * ((gm_dp * (gm_cm + G1)) / (G1 * G2)) ;

  // On essaie de trouver un wt de depart pour gain dynamique = 0. On tatonne en augmentant
  // wt jusqu'a ce que le gain devienne negatif. Les bornes recoltees pour wt
  // doivent etre affinees.
  // wt evolue par decades (1 a 10, puis 10 a 90, puis 100 a 900 etc.)

  gain = fgainWt(K,wt0_Hz * 2.0 * M_PI,w0,w1,w2) ;
  wt_left  = wt0_Hz * 2.0 * M_PI ;
  wt_right = wt0_Hz * 2.0 * M_PI ;

  while(gain > 0.0)
    {
      if(wt0_Hz > 10.0 * decade)
        {
          decade = decade * 10.0 ;
        }

      wt0_Hz = wt0_Hz + decade ;

      wt_left = wt_right ;
      wt_right = wt0_Hz * 2.0 * M_PI ;

      gain = fgainWt(K,wt0_Hz * 2.0 * M_PI,w0,w1,w2) ;
    }

  // On trouve par dichotomie une valeur du gain plus precise
  // wt_left et wt_right sont deja initialisees par la boucle precedente

  while(fabs(wt_left - wt_right) > epsilon)
    {
      wt = (wt_left + wt_right) / 2.0 ;

      gain = fgainWt(K,wt,w0,w1,w2) ;

      // La fonction de gain est decroissante

      if(gain < 0.0)
        {
          wt_right = wt ;
        }
      else
        {
          wt_left = wt ;
        }
```

```
    }

  return wt ;
}

// Frequence de transition en Hz (Unity Gain Frequency)

static double fugf(double wt)
{
  return wt / (2.0 * M_PI) ;
}

// Frequence de coupure en Hz (Frequency cut-off)

static double fcof(double w2)
{
  return w2 / (2.0 * M_PI) ;
}

// Calcul du produit gain-bande en hz
// GBW = gain statique * w2
//     = lineaire * Hz
//
// Gain db ^
//         |                 120 db = 1.06e
//     120 |_________
//         |        :.       GBW = 1.0e6 * 10
//         |        : .          = 1.0e7
//         |        :  .
//         |        :   .
//         |        :    .
//         ---------:------------> F Hz
//                  10

static double fgbw(double gain,double w2)
{
  double gbw = 0.0 ;

  // gain en echelle lineaire (pas d'unite)
  // w2 en rad/s a convertir en Hz
  gbw =  gain * (w2 / (2.0 * M_PI)) ;

  return gbw ;
}

// Calcul de la marge de phase en degres

static double fpm(double wt,double w0,double w1,double w2)
{
  double pm = 0.0 ;

  pm = 180.0 + ((atan2(wt,w0) - atan2(wt,w1) - atan2(wt,w2)) * (180.0 / M_PI)) ;

  return pm ;
}

// Vitesse maximale de variation de la tension de sortie
// Slew-rate en V/s

static double fsr(double ibias,double C2)
{
  double sr = 0.0 ;

  sr = ibias / C2 ;

  return sr ;
}

// Gain statique en db

static double fgain_db(double gm_dp,double gm_cm,double G1,double G2)
{
  double gain = 0.0 ;

  // Facteur multiplicatif constant
  gain = 20.0 * log10(0.5 * ((gm_dp * (gm_cm + G1)) / (G1 * G2))) ;

  return gain ;
}

// Gain statique echelle lineaire

static double fgain_linear(double gm_dp,double gm_cm,double G1,double G2)
{
  double gain = 0.0    ;

  gain = 0.5 * ((gm_dp * (gm_cm + G1)) / (G1 * G2)) ;
```

```
  return gain ;
}


// ---------------------------------------------------------------------------------------------------
// ------------------------------------------- CREATE ------------------------------------------------
// ---------------------------------------------------------------------------------------------------

CAIRO_BEGIN_CREATE(OTA,char *name,char type)

  double l_min  = CAIRO_GET_DOUBLE_TABLE("TRANSISTOR_CMOS_DATA","TR_LENGTH") * MICRON ;
//double l_step = CAIRO_GET_DOUBLE_TABLE("PHYSICAL_GRID") * MICRON ;
  double l_step = CAIRO_GET_DOUBLE_TABLE("PHYSICAL_GRID") * MICRON * 100.0 ;

  CAIRO_SET_LOCAL_VARIABLE("l_min",l_min) ;
  CAIRO_SET_LOCAL_VARIABLE("l_step",l_step) ;

  // ----------------------
  // Parametres structurels
  // ----------------------

  if(CAIRO_CHECK_TRANSISTOR_TYPE(type) == false)
   {
     cpErrorMessage("Unknown transistor type in %s",name) ;
   }

  CAIRO_SET_LOCAL_VARIABLE("type",type) ;

  // --------------
  // Instanciations
  // --------------

  if(IsTransN(type))
   {
     // Miroir de courant simple CM de type P
     CAIRO_CREATE("libMOS","CM_MOS","CM","P3","P4",TRANSP,true) ;

     // Paire differentielle DP de type N
     CAIRO_CREATE("libMOS","DP_MOS","DP","N1","N2",TRANSN,true) ;

     // Transistor simple BIAS de type N
     CAIRO_CREATE("libMOS","TR_MOS","BIAS","N5",TRANSN,true) ;
   }
  else
   {
     if(IsTransP(type))
       {
         // Transistor simple BIAS de type P
         CAIRO_CREATE("libMOS","TR_MOS","BIAS","P5",TRANSP,true) ;

         // Paire differentielle DP de type P
         CAIRO_CREATE("libMOS","DP_MOS","DP","P1","P2",TRANSP,true) ;

         // Miroir de courant simple CM de type N
         CAIRO_CREATE("libMOS","CM_MOS","CM","N3","N4",TRANSN,true) ;
       }
     else
       {
         // Type inconnu
         CAIRO_ERROR_MESSAGE("Unknown transistor type for OTA %s",name) ;
       }
   }

  // ---------------------------
  // Declaration des connecteurs
  // ---------------------------

  if(IsTransN(type))
   {
     CAIRO_IO("VP1" ,CP_WEST ) ; // Tension de polarisation
     CAIRO_IO("VIN1",CP_WEST ) ; // Entree differentielle +
     CAIRO_IO("VIN2",CP_WEST ) ; // Entree differentielle -
     CAIRO_IO("VOUT",CP_EAST ) ; // Sortie
     CAIRO_IO("VDD" ,CP_NORTH) ; // Alimentation
     CAIRO_IO("VSS" ,CP_EAST ) ; // Alimentation
   }
  else
   {
     CAIRO_IO("VP1" ,CP_WEST ) ; // Tension de polarisation
     CAIRO_IO("VIN1",CP_WEST ) ; // Entree differentielle +
     CAIRO_IO("VIN2",CP_WEST ) ; // Entree differentielle -
     CAIRO_IO("VOUT",CP_EAST ) ; // Sortie
     CAIRO_IO("VDD" ,CP_EAST ) ; // Alimentation
     CAIRO_IO("VSS" ,CP_SOUTH) ; // Alimentation
   }

  // ------------------------------------
  // Connexions electriques et conteneurs
```

```
// -----------------------------------

if(IsTransN(type))
 {
   // Attention a l'ordre de VB, VOUT, VIN1 et VIN2 pour la paire differentielle

   // CM : drain1 drain2 source (bulk relie a la source en interne)
   CAIRO_CONNECT("CM","drain1 => VB","drain2 => VOUT","source => VDD");

   // DP : drain1 drain2 gate1 gate2 source (bulk relie a la source en interne)
   CAIRO_CONNECT("DP","drain1 => VB","drain2 => VOUT","gate1 => VIN1","gate2 => VIN2","source => VA") ;

   // BIAS :  drain gate source (bulk relie a la source en interne)

   CAIRO_CONNECT("BIAS","drain => VA","gate => VP1","source => VSS");
   CAIRO_VERTICAL_CONTAINER(name,"BIAS",SYM_Y,SD_NTIE,SD_NTIE,PITCH*2,PITCH*2,
                                "DP"  ,NOSYM,SD_NTIE,2*PITCH,SD_NTIE,SD_NTIE,
                                "CM"  ,NOSYM,2*PITCH,3 * PITCH,SD_NWELL_PTIE,SD_NWELL_PTIE) ;
 }
else
 {
   if(IsTransP(type))
     {
       // Attention a l'ordre de VB, VOUT, VIN1 et VIN2 pour la paire differentielle

       // BIAS : drain gate source (bulk relie a la source en interne)
       CAIRO_CONNECT("BIAS","drain => VA","gate => VP1","source => VDD");

       // DP : drain1 drain2 gate1 gate2 source (bulk relie a la source en interne)
       CAIRO_CONNECT("DP","drain1 => VOUT","drain2 => VB","gate1 => VIN2","gate2 => VIN1","source => VA") ;

       // CM : drain1 drain2 source (bulk relie a la source en interne)
       CAIRO_CONNECT("CM","drain1 => VB","drain2 => VOUT","source => VSS");

       CAIRO_VERTICAL_CONTAINER(name,"CM"  ,NOSYM,3 * PITCH,2 * PITCH,SD_NWELL_PTIE,SD_NWELL_PTIE,
                                     "DP"  ,NOSYM,2 * PITCH,SD_NTIE  ,SD_NTIE      ,SD_NTIE      ,
                                     "BIAS",NOSYM,2 * PITCH,SD_NTIE  ,PITCH * 2    ,PITCH * 2     ) ;
     }
 }

// --------------------------
// Declaration des parametres
// --------------------------


CAIRO_DECLARE_PARAM("TEMP"    ) ;
CAIRO_DECLARE_PARAM("VEG_BIAS") ;
CAIRO_DECLARE_PARAM("VEG_DP"  ) ;
CAIRO_DECLARE_PARAM("L_INIT"  ) ;
CAIRO_DECLARE_PARAM("VIN"     ) ;
CAIRO_DECLARE_PARAM("VOUT"    ) ;

CAIRO_DECLARE_PARAM("VP1"      ) ;
CAIRO_DECLARE_PARAM("UGF"      ,CP_PRIMARY) ;
CAIRO_DECLARE_PARAM("PM"       ,CP_PRIMARY) ;
CAIRO_DECLARE_PARAM("GBW"      ) ;
CAIRO_DECLARE_PARAM("COF"      ) ;
CAIRO_DECLARE_PARAM("SR"       ) ;
CAIRO_DECLARE_PARAM("GAIN"     ) ;
CAIRO_DECLARE_PARAM("CLOAD"    ) ;
CAIRO_DECLARE_PARAM("CLOAD_OUT") ;

CAIRO_DECLARE_PARAM("IBIAS") ;


// --------------------------
// Declaration des procedures
// --------------------------


CAIRO_DECLARE_PROCEDURE("VP1",CP_UNCOMPLETE,"TEMP",CP_IN,"VEG_BIAS",CP_IN,"VEG_DP",CP_IN,
                                          "L_INIT",CP_IN,"VIN",CP_IN,"VOUT",CP_IN,"IBIAS",CP_INOUT,"VP1",CP_OUT) ;

CAIRO_DECLARE_PROCEDURE("UGF",CP_COMPLETE,"TEMP",CP_IN,"CLOAD",CP_IN,"UGF",CP_OUT,"PM",CP_OUT,"CLOAD_OUT",CP_OUT,"IBIAS",CP_INOUT,
                                        "SR",CP_OUT,"GAIN",CP_OUT,"COF",CP_OUT,"GBW",CP_OUT) ;

CAIRO_DECLARE_PROCEDURE("UGF_MIN_PM",CP_COMPLETE,"TEMP",CP_IN,"CLOAD",CP_IN,"UGF",CP_OUT,"PM",CP_OUT,"CLOAD_OUT",CP_OUT,
                                             "IBIAS",CP_INOUT,"SR",CP_OUT,"GAIN",CP_OUT,"COF",CP_OUT,"GBW",CP_OUT) ;

// ------------------------
// Declaration des messages
// ------------------------


// Messages de VP1
CAIRO_DECLARE_PROCEDURE_MESSAGE(1 ,"VP1: DP: IBIAS is too large, try to decrease current") ;
CAIRO_DECLARE_PROCEDURE_MESSAGE(2 ,"VP1: DP: IBIAS is too small, try to increase current") ;
CAIRO_DECLARE_PROCEDURE_MESSAGE(3 ,"VP1: DP: W of DP is too small, try to increase current") ;
```

```
  CAIRO_DECLARE_PROCEDURE_MESSAGE(4 ,"VP1: BIAS: IBIAS is too large, try to decrease current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(5 ,"VP1: BIAS: IBIAS is too small, try to increase current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(6 ,"VP1: BIAS: W of DP is too small, try to increase current") ;

  CAIRO_DECLARE_PROCEDURE_MESSAGE(7 ,"VP1: CM: IBIAS is too large, try to decrease current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(8 ,"VP1: CM: IBIAS is too small, try to increase current") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(9 ,"VP1: CM: W of CM is too small, try to increase current") ;

  // Messages de UGF
  CAIRO_DECLARE_PROCEDURE_MESSAGE(10,"UGF: IBIAS is too large, try to decrease current or UGF or increase PM") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(11,"UGF: IBIAS is too small, try to increase current or UGF or decrease PM") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(12,"UGF: IBIAS is too small because W is too small, try to increase current or UGF or decrease PM") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(13,"UGF: IBIAS is too large in PM loop (L too small), try to decrease current or UGF or increase PM") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(14,"UGF: IBIAS too small, SR too small, SR unsatisfied") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(15,"UGF: IBIAS too small, GAIN too small, GAIN unsatisfied") ;

  CAIRO_DECLARE_PROCEDURE_MESSAGE(16,"VP1: BIAS: transistor mode is cut off, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(17,"VP1: BIAS: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(18,"VP1: BIAS: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(19,"VP1: BIAS: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(20,"VP1: BIAS: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(21,"VP1: DP: transistor mode is cut off, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(22,"VP1: DP: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(23,"VP1: DP: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(24,"VP1: DP: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(25,"VP1: DP: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(26,"VP1: CM: transistor mode is cut off, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(27,"VP1: CM: transistor mode is subthreshold, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(28,"VP1: CM: transistor mode is strong inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(29,"VP1: CM: transistor mode is weak inversion & linear, decrease VEG (VDS should be over VEG)") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(30,"VP1: CM: transistor mode is weak inversion & saturation, decrease VEG (VDS should be over VEG)") ;

CAIRO_END_CREATE(OTA)


// ------------------------------------------------------------------------------------------------
// ---------------------------------------------- DSES ----------------------------------------------
// ------------------------------------------------------------------------------------------------

CAIRO_BEGIN_DSES(OTA,char *name)

  char   type         = TRANSN ;

  double temp         = 0.0 ; // Specifications
  double vdd          = 0.0 ;
  double vss          = 0.0 ;
  double veg_bias     = 0.0 ;
  double veg_dp       = 0.0 ;
  double l_init       = 0.0 ;
  double l_min        = 0.0 ;
  double l_step       = 0.0 ;
  double vin          = 0.0 ; // Tension de mode commun
  double vout         = 0.0 ; // Tension de mode commun
  double ibias_in     = 0.0 ; // Courtant de polarisation
  double ibias_cal    = 0.0 ; // Courtant de polarisation calcule
  double cload_in     = 0.0 ; // Capacite de charge entrante
  double cload_max    = 0.0 ; // Contrainte sur cload
  double cload_cal    = 0.0 ; // Somme des capacites parasites de routage et de CLOAD
  double cpar         = 0.0 ; // Capacites de routage
  double vp1          = 0.0 ; // Tension de polaridation
  double ugf_in       = 0.0 ; // Contrainte sur ugf (unity Gain Frequency, frequence de transition) en Hz
  double ugf_cal      = 0.0 ; // ugf calcule en Hz
  double ugf_in_mhz   = 0.0 ; // Contrainte sur ugf en MHz (pour les tests)
  double ugf_cal_mhz  = 0.0 ; // ugf calcule en MHz (pour les tests)
  double pm_in_low    = 0.0 ; // Contrainte sur la marge de phase borne basse (en degre)
  double pm_in_high   = 0.0 ; // Contrainte sur la marge de phase borne haute (en degre)
  double pm_cal       = 0.0 ; // Marge de phase calculee (en degre)
  double gbw_cal      = 0.0 ; // Produit gain-bande en Hz
  double cof_cal      = 0.0 ; // cof (frequence de coupure, frequency cut-off) calculee en Hz
  double sr_in        = 0.0 ; // Vitesse maximale de variation de la tension de sortie (Slew-rate)
  double sr_cal       = 0.0 ;
  double gain_in      = 0.0 ; // Gain statique en db
  double gain_cal_db  = 0.0 ; // Gain statique en db
  double gain_cal_lin = 0.0 ; // Gain statique lineaire
  double vgs_bias     = 0.0 ;
  double vbs_bias     = 0.0 ;
  double vds_bias     = 0.0 ;
  double gds_bias     = 0.0 ;
  double ids_bias     = 0.0 ;
  double w_bias       = 0.0 ;
  double l_bias       = 0.0 ;
  string mode_bias          ;

  double vth_dp       = 0.0 ;
  double vgs_dp       = 0.0 ;
  double vbs_dp       = 0.0 ;
  double vds_dp       = 0.0 ;
```

```
double ids_dp       = 0.0 ;
double w_dp         = 0.0 ;
double l_dp         = 0.0 ;
double cgd_dp       = 0.0 ;
double cbd_dp       = 0.0 ;
double gm_dp        = 0.0 ;
double gds_dp       = 0.0 ;
double gmb_dp       = 0.0 ;
string mode_dp            ;

double vgs_cm       = 0.0 ;
double vds_cm       = 0.0 ;
double ids_cm       = 0.0 ;
double w_cm         = 0.0 ;
double l_cm         = 0.0 ;
double cgs_cm       = 0.0 ;
double cgb_cm       = 0.0 ;
double cbd_cm       = 0.0 ;
double cgd_cm       = 0.0 ;
double gm_cm        = 0.0 ;
double gds_cm       = 0.0 ;
string mode_cm            ;

double wt           = 0.0 ; // Pulsation au point de transition en rad/s
double w0           = 0.0 ; // Pulsation du zero en rad/s
double w1           = 0.0 ; // Pulsation du premier pole non dominant en rad/s
double w2           = 0.0 ; // Pulsation du second pole non dominant en rad/s
double G1           = 0.0 ;
double G2           = 0.0 ;
double C2           = 0.0 ;

double va           = 0.0 ;
double va_next      = 0.0 ;

double epsilon      = 0.001 ; // Variation de 0.1 %% (1 pour mille)
double epsilon2     = 0.15  ; // Variation de 15 % (15 pour 100)

double sign         = 1.0 ;

bool   vp1_called            = false ;
bool   objective_minimize_pm = false ;
bool   minimize_pm           = false ;
bool   pm_satisfied          = false ;
bool   ugf_satisfied         = false ;

bool   loop_minimize         = false ;
bool   loop_within           = false ;
bool   loop_ugf              = false ;
bool   pm_too_small          = false ;
bool   pm_too_large          = false ;
bool   error_in_pm           = false ;
double l_low                 = 0.0   ;
double l_high                = 0.0   ;
double l_max                 = 1.0   ; // 1 metre (nombre magique !)
bool   constraint_pm_within  = false ;
double delta_ugf             = 0.0   ;

CAIRO_GET_LOCAL_VARIABLE("type",type) ;
CAIRO_GET_LOCAL_VARIABLE("l_min",l_min) ;
CAIRO_GET_LOCAL_VARIABLE("l_step",l_step) ;

l_low = l_min ;
l_high = l_max ;

if(IsTransP(type))
 {
   sign = -1.0 ;
 }

// -----------------------
// Parametres obligatoires
// -----------------------

// temp
CAIRO_TRY_GET_VALUE("TEMP",temp)
IF_NO_VALUE
   FATAL_ERROR_PARAM("TEMP","parameter not set",LOCATION);
ENDIF_NO_VALUE

// vdd
CAIRO_GET_GLOBAL_VARIABLE("VDD",vdd) ;

// vss
CAIRO_GET_GLOBAL_VARIABLE("VSS",vss) ;

// -----------------
// ------------- VP1
// -----------------
```

```
// La procedure VP1 peut etre appelee seule (depuis le niveau hierarchique
// superieur), ou depuis le niveau courant par la procedure UGF
// Il faut donc prevoir l'appel interne (certaines variables locales
// et certains parametres sont alors positionnes par UGF)

CAIRO_BEGIN_PROCEDURE("VP1")

  CAIRO_SET_LOCAL_VARIABLE("vp1_called",true) ;

  // IBIAS peut venir du niveau superieur ou avoir ete positionne par la
  // procedure UGF (communication interne par parametres)

  CAIRO_TRY_GET_VALUE("IBIAS",ibias_in)
  IF_NO_VALUE
    CAIRO_ERROR_MESSAGE("Missing input IBIAS in %s when calling VP1",name) ;
  ENDIF_NO_VALUE

  CAIRO_TRY_GET_VALUE("VIN",vin)
  IF_NO_VALUE
    vin = vdd / 2.0 ;
  ENDIF_NO_VALUE

  // vout
  CAIRO_TRY_GET_VALUE("VOUT",vout)
  IF_NO_VALUE
    vout = vdd / 2.0 ;
  ENDIF_NO_VALUE

  // veg_bias
  CAIRO_TRY_GET_VALUE("VEG_BIAS",veg_bias)
    veg_bias = sign * veg_bias ;
  IF_NO_VALUE
    veg_bias = sign * 0.5 ;
  ENDIF_NO_VALUE

  // veg_dp
  CAIRO_TRY_GET_VALUE("VEG_DP",veg_dp)
    veg_dp = sign * veg_dp ;
  IF_NO_VALUE
    veg_dp = sign * 0.2 ;
  ENDIF_NO_VALUE

  // l_init
  CAIRO_TRY_GET_VALUE("L_INIT",l_init)
  IF_NO_VALUE
    l_init = l_min ;
  ENDIF_NO_VALUE

  ids_dp = sign * ibias_in ;
  ids_cm = - sign * (ibias_in / 2.0) ;

  // Si les L ne sont pas positionnes, on prend les L memorises
  // car ils viennent de la procedure UGF. Sinon les L
  // sont initialises au L minimum de la technologie

  CAIRO_TRY_GET_LOCAL_VARIABLE("l_bias",l_bias)
  IF_NO_LOCAL_VARIABLE
    l_bias = l_init ;
  ENDIF_NO_VALUE

  CAIRO_TRY_GET_LOCAL_VARIABLE("l_dp",l_dp)
  IF_NO_LOCAL_VARIABLE
    l_dp = l_init ;
  ENDIF_NO_VALUE

  CAIRO_TRY_GET_LOCAL_VARIABLE("l_cm",l_cm)
  IF_NO_LOCAL_VARIABLE
    l_cm = l_init ;
  ENDIF_NO_VALUE

  if(IsTransP(type))
   {
     // On doit avoir vout < va < vdd
     va = vdd + 2.0 * veg_bias ;

     // Securite
     if((va < vout) || (va > vdd))
       {
         va = fabs((vdd + vout) / 2.0) ;
       }
   }
  else
   {
     // On doit avoir vss < va < vout
     va = vout - 2.0 * veg_bias ;

     // Securite
```

```
   if((va < vss) || (va > vout))
     {
       va = fabs((vout + vss) / 2.0) ;
     }
 }

va_next = va ;

CAIRO_SET_PARAM("DP","TEMP",temp) ;
CAIRO_SET_PARAM("DP","VBS",vbs_dp) ;
CAIRO_SET_PARAM("DP","VEG",veg_dp) ;
CAIRO_SET_PARAM("DP","L",l_dp) ;
CAIRO_SET_PARAM("DP","IBIAS",ids_dp) ;

do
  {
    va = va_next ;

    vds_dp = vout - va ;

    CAIRO_SET_PARAM("DP","VDS",vds_dp) ;

    CAIRO_COMPUTE("DP","W(VEG,L,IBIAS)") ;

    ON_PROCEDURE_STATUS
      switch(PROCEDURE_STATUS)
        {
          case 10 : CAIRO_SET_RETURN_PROCEDURE(1) ;
                    CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_LARGE) ;
                    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                    goto return_vp1 ;
                    break ;
          case 11 : CAIRO_SET_RETURN_PROCEDURE(2) ;
                    CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                    goto return_vp1 ;
                    break ;
          case 12 : CAIRO_SET_RETURN_PROCEDURE(3) ;
                    CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                    goto return_vp1 ;
                    break ;
          default : CAIRO_ERROR_MESSAGE("Unknown status %li for DP:W(VEG,L,IBIAS) in VP1\n",PROCEDURE_STATUS) ;
                    break ;
        }
    END_ON_PROCEDURE_STATUS

    CAIRO_GET_PARAM("DP","VTH",vth_dp) ;

    va_next = vin - vth_dp - veg_dp ;
  } while(fabs(va_next - va) > epsilon) ;

// ----- transistor BIAS

if(IsTransP(type))
 {
   vds_bias = va - vdd ;
 }
else
 {
   vds_bias = va ;
 }

ids_bias = sign * ibias_in ;

CAIRO_SET_PARAM("BIAS","TEMP",temp) ;
CAIRO_SET_PARAM("BIAS","VDS",vds_bias) ;
CAIRO_SET_PARAM("BIAS","VBS",vbs_bias) ;
CAIRO_SET_PARAM("BIAS","VEG",veg_bias) ;
CAIRO_SET_PARAM("BIAS","L",l_bias) ;
CAIRO_SET_PARAM("BIAS","IDS",ids_bias) ;

CAIRO_COMPUTE("BIAS","W(VEG,L,IDS)") ;

ON_PROCEDURE_STATUS
  switch(PROCEDURE_STATUS)
    {
      case 10 : CAIRO_SET_RETURN_PROCEDURE(4) ;
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                goto return_vp1 ;
                break ;
      case 11 : CAIRO_SET_RETURN_PROCEDURE(5) ;
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                goto return_vp1 ;
                break ;
      case 12 : CAIRO_SET_RETURN_PROCEDURE(6) ;
```

```
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                goto return_vp1 ;
                break ;
      default : CAIRO_ERROR_MESSAGE("Unknown status %li for BIAS:W(VEG,L,IBIAS) in VP1\n",PROCEDURE_STATUS) ;
                break ;
    }

END_ON_PROCEDURE_STATUS

// Parametres petits signaux
CAIRO_GET_PARAM("BIAS","VGS",vgs_bias) ;
CAIRO_GET_PARAM("BIAS","GDS",gds_bias) ;

// Positionnement pour la procedure UGF
CAIRO_SET_LOCAL_VARIABLE("gds_bias",gds_bias) ;

// ----- Paire differentielle DP
vgs_dp = vin - va ;

// Parametres petits signaux
CAIRO_GET_PARAM("DP","CGD",cgd_dp) ;
CAIRO_GET_PARAM("DP","CBD",cbd_dp) ;
CAIRO_GET_PARAM("DP","GM" ,gm_dp)  ;
CAIRO_GET_PARAM("DP","GDS",gds_dp) ;
CAIRO_GET_PARAM("DP","GMB",gmb_dp) ;

// Positionnement pour la procedure UGF
CAIRO_SET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
CAIRO_SET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;
CAIRO_SET_LOCAL_VARIABLE("gm_dp" ,gm_dp)  ;
CAIRO_SET_LOCAL_VARIABLE("gds_dp",gds_dp) ;
CAIRO_SET_LOCAL_VARIABLE("gmb_dp",gmb_dp) ;

// ----- Miroir de courant CM

if(IsTransP(type))
 {
   vds_cm = vout ;
 }
else
 {
   vds_cm = vout - vdd ;
 }

vgs_cm = vds_cm ;

// Parametres petits signaux
// Aucune fonction de la calculette n'ayant ete appelee, on choisit W(L,IREF)
CAIRO_SET_PARAM("CM","TEMP",temp)   ;
CAIRO_SET_PARAM("CM","VDS",vds_cm)  ;
CAIRO_SET_PARAM("CM","L",l_cm)      ;
CAIRO_SET_PARAM("CM","IREF",ids_cm) ;

CAIRO_COMPUTE("CM","W(L,IREF)")     ;

ON_PROCEDURE_STATUS
  switch(PROCEDURE_STATUS)
    {
      case 4  : CAIRO_SET_RETURN_PROCEDURE(7) ;
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                goto return_vp1 ;
                break ;
      case 5  : CAIRO_SET_RETURN_PROCEDURE(8) ;
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
                goto return_vp1 ;
                break ;
      case 6  : CAIRO_SET_RETURN_PROCEDURE(9) ;
                CAIRO_SET_VALUE("IBIAS",ibias_in,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;

                goto return_vp1 ;
                break ;
      default : CAIRO_ERROR_MESSAGE("Unknown status %li for CM:W(VEG,L,IBIAS) in VP1\n",PROCEDURE_STATUS) ;
                break ;

    }
END_ON_PROCEDURE_STATUS

CAIRO_GET_PARAM("CM","CGS",cgs_cm) ;
CAIRO_GET_PARAM("CM","CGB",cgb_cm) ;
CAIRO_GET_PARAM("CM","CBD",cbd_cm) ;
CAIRO_GET_PARAM("CM","CGD",cgd_cm) ;
CAIRO_GET_PARAM("CM","GM" ,gm_cm)  ;
CAIRO_GET_PARAM("CM","GDS",gds_cm) ;
```

```
// Positionnement pour la procedure UGF
CAIRO_SET_LOCAL_VARIABLE("cgs_cm",cgs_cm) ;
CAIRO_SET_LOCAL_VARIABLE("cgb_cm",cgb_cm) ;
CAIRO_SET_LOCAL_VARIABLE("cbd_cm",cbd_cm) ;
CAIRO_SET_LOCAL_VARIABLE("cgd_cm",cgd_cm) ;
CAIRO_SET_LOCAL_VARIABLE("gm_cm" ,gm_cm ) ;
CAIRO_SET_LOCAL_VARIABLE("gds_cm",gds_cm) ;

CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

// ----- vp1

if(IsTransP(type))
 {
   vp1 = vdd + vgs_bias ;
 }
else
 {
   vp1 = vgs_bias ;
 }

// Verification du mode de fonctionnement des transistors
// Ils doivent tous etre en saturation

CAIRO_GET_PARAM("BIAS","MODE",mode_bias) ;
CAIRO_GET_PARAM("DP"  ,"MODE",mode_dp  ) ;
CAIRO_GET_PARAM("CM"  ,"MODE",mode_cm  ) ;

if(mode_bias == "cut off")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(16) ;
    goto return_vp1 ;
  }
else
if(mode_bias == "subthreshold")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(17) ;
    goto return_vp1 ;
  }
else
if(mode_bias == "strong inversion & linear")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(18) ;
    goto return_vp1 ;
  }
else
if(mode_bias == "weak inversion & linear")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(19) ;
    goto return_vp1 ;
  }
else
if(mode_bias == "weak inversion & saturation")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(20) ;
    goto return_vp1 ;
  }
else
if(mode_bias == "Unknown mode")
  {
    CAIRO_ERROR_MESSAGE("Unknown mode for BIAS in VP1 for OTA %s",name) ;
  }

if(mode_dp == "cut off")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(21) ;
    goto return_vp1 ;
  }
else
if(mode_dp == "subthreshold")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(22) ;
    goto return_vp1 ;
  }
else
if(mode_dp == "strong inversion & linear")
  {
    CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
    CAIRO_SET_RETURN_PROCEDURE(23) ;
```

```
        goto return_vp1 ;
      }
    else
    if(mode_dp == "weak inversion & linear")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(24) ;
        goto return_vp1 ;
      }
    else
    if(mode_dp == "weak inversion & saturation")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(25) ;
        goto return_vp1 ;
      }
    else
    if(mode_dp == "Unknown mode")
      {
        CAIRO_ERROR_MESSAGE("Unknown mode for DP in VP1 for OTA %s",name) ;
      }

    if(mode_cm == "cut off")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(26) ;
        goto return_vp1 ;
      }
    else
    if(mode_cm == "subthreshold")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(27) ;
        goto return_vp1 ;
      }
    else
    if(mode_cm == "strong inversion & linear")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(28) ;
        goto return_vp1 ;
      }
    else
    if(mode_cm == "weak inversion & linear")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(29) ;
        goto return_vp1 ;
      }
    else
    if(mode_cm == "weak inversion & saturation")
      {
        CAIRO_SET_VALUE("VP1",vp1,CP_NAV) ;
        CAIRO_SET_RETURN_PROCEDURE(30) ;
        goto return_vp1 ;
      }
    else
    if(mode_cm == "Unknown mode")
      {
        CAIRO_ERROR_MESSAGE("Unknown mode for CM in VP1 for OTA %s",name) ;
      }

    CAIRO_SET_VALUE("VP1",vp1,CP_VALID) ;
    CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;

return_vp1 :
    return ;

  END_PROCEDURE

  // ----------------------------------------
  // ------------ UGF sans minimisation de PM
  // ----------------------------------------

  // La procedure UGF appelle en interne la procedure VP1 pour le calcul des parametres
  // petits signaux (point de polarisation).
  // Les valeurs des L sont passees par des variables locales, la valeur de IBIAS par
  // parametre. Les capacites sont passees par variables locales.

  CAIRO_BEGIN_PROCEDURE("UGF")

    // Capacite de charge
    CAIRO_TRY_GET_VALUE("CLOAD",cload_in)
      // Rappeler les parasites de routage
      // et en tenir compte
      cload_cal = cload_in + cpar ;
    IF_NO_VALUE
      CAIRO_ERROR_MESSAGE("Missing input CLOAD in %s in UGF",name) ;
```

```
ENDIF_NO_VALUE

// ------------------ Contraintes

// Contrainte sur UGF
EXAMINE_CONSTRAINT("UGF")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("UGF",ugf_in) ;
                            ugf_in_mhz = ceil(ugf_in / 1.0e6) ;
                            break ;
  CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on UGF in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for UGF in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Contrainte sur PM
EXAMINE_CONSTRAINT("PM")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("PM",pm_in_low) ;
                            pm_in_high = 180 ;
                            break ;
  CASE_WITHIN             : GET_CONSTRAINT_ASSOCIATED_VALUES("PM",pm_in_low,pm_in_high) ;
                            constraint_pm_within = true ;
                            break ;
  CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on PM in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for PM in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Contrainte sur CLOAD_OUT
EXAMINE_CONSTRAINT("CLOAD_OUT")
  CASE_BELOW              : GET_CONSTRAINT_ASSOCIATED_VALUES("CLOAD_OUT",cload_max) ;
                            break ;
  CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on CLOAD_OUT in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for CLOAD_OUT in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE


// On rappel VP1 pour obtenir les parametres petits signaux
// C'est aussi necessaire pour la boucle de stabilite (sinon VP1 n'est jamais
// rappele car ugf est satisfait, l n'est pas modifie)

CAIRO_TRY_GET_LOCAL_VARIABLE("vp1_called",vp1_called)
IF_NO_LOCAL_VARIABLE
  RESET_PROCEDURE_STATUS ;

  while(PROCEDURE_STATUS != CP_OK)
    {
      CAIRO_COMPUTE("VP1") ;

      ON_PROCEDURE_STATUS

        switch(PROCEDURE_STATUS)
          {
            case 1  :
            case 4  :
            case 7  : CAIRO_SET_RETURN_PROCEDURE(10) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_LARGE) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_LARGE) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf ;
                      break ;

            case 2  :
            case 5  :
            case 8  : CAIRO_SET_RETURN_PROCEDURE(11) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_LARGE) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf ;
                      break ;

            case 3  :
            case 6  :
            case 9  : l_bias = l_bias + l_step ;
                      l_dp = l_dp + l_step ;
                      l_cm = l_cm + l_step ;
                      CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;
```

```
                    break ;

            case 15 :
            case 16 :
            case 17 :
            case 18 :
            case 19 :
            case 20 :
            case 21 :
            case 22 :
            case 23 :
            case 24 :
            case 25 :
            case 26 :
            case 27 :
            case 28 :
            case 29 : CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf ;
                      break ;

            default : CAIRO_ERROR_MESSAGE("Unknown status %li for VP1,IBIAS\n",PROCEDURE_STATUS) ;
                      break ;
          }

      END_ON_PROCEDURE_STATUS
    }

  CAIRO_DELETE_LOCAL_VARIABLE("vp1_called") ;
ENDIF_NO_LOCAL_VARIABLE

// Parametres essentiels manquants issus de la polarisation
// Passage de parametres par variables locales

CAIRO_TRY_GET_VALUE("IBIAS",ibias_cal)
IF_NO_VALUE
  CAIRO_ERROR_MESSAGE("Missing input IBIAS in %s when calling VP1",name) ;
ENDIF_NO_VALUE

CAIRO_GET_LOCAL_VARIABLE("gds_bias",gds_bias) ;
CAIRO_GET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
CAIRO_GET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;
CAIRO_GET_LOCAL_VARIABLE("gm_dp" ,gm_dp ) ;
CAIRO_GET_LOCAL_VARIABLE("gds_dp",gds_dp) ;
CAIRO_GET_LOCAL_VARIABLE("gmb_dp",gmb_dp) ;
CAIRO_GET_LOCAL_VARIABLE("cgs_cm",cgs_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cgb_cm",cgb_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cbd_cm",cbd_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cgd_cm",cgd_cm) ;
CAIRO_GET_LOCAL_VARIABLE("gm_cm" ,gm_cm ) ;
CAIRO_GET_LOCAL_VARIABLE("gds_cm",gds_cm) ;

CAIRO_GET_LOCAL_VARIABLE("l_bias",l_bias) ;
CAIRO_GET_LOCAL_VARIABLE("l_dp",l_dp) ;
CAIRO_GET_LOCAL_VARIABLE("l_cm",l_cm) ;

// Initialisation: on se base sur les donnees du point de polarisation

ids_bias = sign * ibias_cal ;
ids_dp = sign * ibias_cal ;
ids_cm = - sign * (ibias_cal / 2.0) ;

// Calcul des parametres intermediaires
fw0w1w2G1G2C2(gds_dp,cgd_dp,cbd_dp,
              gds_cm,gm_cm,cgs_cm,cgb_cm,cbd_cm,cgd_cm,
              cload_cal,
              w0,w1,w2,G1,G2,C2) ;

// Calcul de la pulsation au point de transition
wt = fwt(gm_dp,gm_cm,G1,G2,w0,w1,w2,C2) ;
ugf_cal = fugf(wt) ;

// Gain statique echelle lineaire
gain_cal_lin = fgain_linear(gm_dp,gm_cm,G1,G2) ;

// Produit gain-bande
gbw_cal = fgbw(gain_cal_lin,w2) ;

// Marge de phase
pm_cal = fpm(wt,w0,w1,w2) ;

// Frequence de coupure
cof_cal = fcof(w2) ;
```

```
do
  {
    // ***** Boucle sur UGF

    loop_compute_ugf :

    loop_ugf      = false ;
    loop_within   = false ;
    pm_satisfied  = false ;
    pm_too_small  = false ;
    pm_too_large  = false ;
    error_in_pm   = false ;

    do
      {
        // ***** Boucle sur PM

        loop_within_pm :

        // --- Boucle pour satisfaire pm_low <= pm_cal

        if(pm_cal < pm_in_low)
          {
            loop_within = true ;
            pm_too_small = true ;
            pm_too_large  = false ;

            l_bias = l_bias - l_step ;
            l_dp = l_dp - l_step ;
            l_cm = l_cm - l_step ;
            l_low = l_bias ;

            CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
            CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
            CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

            if(l_bias < l_min)
              {
                CAIRO_SET_RETURN_PROCEDURE(12) ;
                CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                goto bad_return_pm_ugf ;
              }

            goto compute_pm_ugf ;
          }
        else
          {
            if((constraint_pm_within == true) && (pm_in_high < pm_cal))
              {
                // On a pm_high <= pm_cal

                loop_within = true ;
                pm_too_small = false ;
                pm_too_large = true ;

                l_bias = l_bias + l_step ;
                l_dp = l_dp + l_step ;
                l_cm = l_cm + l_step ;
                l_high = l_bias ;

                CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

                goto compute_pm_ugf ;
              }
            else
              {
                // On a pm_low <= pm_cal ou pm_low <= pm_cal << pm_high

                loop_within = false ;
                pm_too_small = false ;
                pm_too_large = false ;

                // Si l'on ne doit pas minimiser pm, alors pm est satisfait
                pm_satisfied = true ;
              }
          }
      }while(pm_satisfied == false) ;


    // --- Boucle pour satisfaire ugf
```

```
    delta_ugf = fabs((ugf_in - ugf_cal) / ugf_cal) ;

    // Inutile d'avoir une precision trop grande, ce qui peut poser des problemes
    // de convergence
    ugf_cal_mhz = ugf_cal / 1.0e6 ;

    if((ugf_cal_mhz < ugf_in_mhz) || (delta_ugf > epsilon2))
      {
        // Aucun contrainte n'est satisfaite (ni ugf >= ugf_in, ni ugf = ugf_in, moyennant epsilon2)

        loop_ugf = true ;
        l_bias = l_min ;
        l_dp = l_min ;
        l_cm = l_min ;

        CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
        CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
        CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

        ibias_cal = ibias_cal * (ugf_in / ugf_cal) ; // ugf_in / ugf_cal > 1

        CAIRO_SET_PARAM("IBIAS",ibias_cal) ;

        goto compute_pm_ugf ;
      }
    else
      {
        // on a ugf >= ugf_in, on arrete
        loop_ugf = false ;
        ugf_satisfied = true ;
      }
  }while(ugf_satisfied == false) ;

// Boucle sur le SR (contrainte additionnelle)

sr_cal = fsr(ibias_cal,C2) ;

EXAMINE_CONSTRAINT("SR")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("SR",sr_in) ;

                            if(sr_cal >= sr_in)
                              {
                                CAIRO_SET_VALUE("SR",sr_cal,CP_VALID) ;
                              }
                            else
                              {
                                CAIRO_SET_RETURN_PROCEDURE(13) ;
                                CAIRO_SET_VALUE("SR",sr_cal,CP_TOO_SMALL) ;
                              }
                            break ;
  CASE_NONE               : CAIRO_SET_VALUE("SR",sr_cal,CP_VALID) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for SR in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Boucle sur le GAIN (contrainte additionnelle)

gain_cal_db = fgain_db(gm_dp,gm_cm,G1,G2) ;

EXAMINE_CONSTRAINT("GAIN")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("GAIN",gain_in) ;

                            if(gain_cal_db >= gain_in)
                              {
                                CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_VALID) ;
                              }
                            else
                              {
                                CAIRO_SET_RETURN_PROCEDURE(14) ;
                                CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_TOO_SMALL) ;
                              }
                            break ;
  CASE_NONE               : CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_VALID) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for GAIN in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

CAIRO_SET_VALUE("UGF",ugf_cal,CP_VALID) ;
CAIRO_SET_VALUE("PM",pm_cal,CP_VALID) ;
CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_VALID) ;
CAIRO_SET_VALUE("CLOAD_OUT",cload_cal,CP_VALID) ;
CAIRO_SET_VALUE("COF",cof_cal,CP_VALID) ;
CAIRO_SET_VALUE("GBW",gbw_cal,CP_VALID) ;

// Appel des procedures completes des fils
```

```
CAIRO_COMPUTE("BIAS","direct_size") ;
CAIRO_COMPUTE("DP","direct_size") ;
CAIRO_COMPUTE("CM","direct_size") ;

CAIRO_GET_PARAM("BIAS","W",w_bias) ;
CAIRO_GET_PARAM("DP","W",w_dp) ;
CAIRO_GET_PARAM("CM","W",w_cm) ;

ids_bias = sign * ibias_cal ;
ids_dp = sign * ibias_cal ;
ids_cm = - sign * (ibias_cal / 2.0) ;

CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;

return ;


// ------------------------------------------------------------
// ---------------------------------- Calculs sur PM et UGF
// ------------------------------------------------------------

compute_pm_ugf :

// --- Calcul de la polarisation

CAIRO_TRY_GET_LOCAL_VARIABLE("vp1_called",vp1_called)

  // La polarisation a deja ete appelee separement, on se contente de rappeler les
  // variables memorisees

  CAIRO_GET_LOCAL_VARIABLE("gds_bias",gds_bias) ;

  CAIRO_GET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("gm_dp" ,gm_dp ) ;
  CAIRO_GET_LOCAL_VARIABLE("gds_dp",gds_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("gmb_dp",gmb_dp) ;

  CAIRO_GET_LOCAL_VARIABLE("cgs_cm",cgs_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cgb_cm",cgb_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cbd_cm",cbd_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cgd_cm",cgd_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("gm_cm" ,gm_cm ) ;
  CAIRO_GET_LOCAL_VARIABLE("gds_cm",gds_cm) ;

  CAIRO_DELETE_LOCAL_VARIABLE("vp1_called") ;

IF_NO_LOCAL_VARIABLE

  // Si la polarisation n'a pas deja ete appelee separement ou que l'on
  // se trouve deja a l'iteration > 1, on l'appelle en interne
  // Les parametres ont etes positionnes. Attention: il se peut que IBIAS
  // doivent etre repositionne (l'iteration precedente a peut-etre echoue.
  // Dans ce cas, IBIAS garde le statut echec)

  RESET_PROCEDURE_STATUS ;

  while(PROCEDURE_STATUS != CP_OK)
    {
      CAIRO_COMPUTE("VP1") ;

      ON_PROCEDURE_STATUS

        switch(PROCEDURE_STATUS)
          {
            case 1  :
            case 4  :
            case 7  : if((pm_too_small == false) && (pm_too_large == false) && (loop_ugf == false))
                        {
                          // On est dans l'intervalle de PM, mais on a trop fait augmenter L (pour
                          // minimiser PM). On a trouve une borne superieure pour L qui reduit
                          // l'intervalle de PM. Il faut revenir sur ses pas

                          error_in_pm = true ;

                          l_high = l_bias ;
                          l_bias = l_bias - l_step ;
                          l_dp = l_dp - l_step ;
                          l_cm = l_cm - l_step ;

                          CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                          CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                          CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

                          CAIRO_SET_PARAM("IBIAS",ibias_cal) ;

                          // Utilisation de la boucle pour aller a l'etiquette compute_pm_ugf
                        }
```

```
                    else
                      {
                        CAIRO_SET_RETURN_PROCEDURE(10) ;
                        CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_LARGE) ;
                        CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_SMALL) ;
                        CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_LARGE) ;
                        CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                        CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                        goto bad_return_pm_ugf ;
                      }

                    break ;

            case 2  :
            case 5  :
            case 8  : CAIRO_SET_RETURN_PROCEDURE(11) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_LARGE) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf ;
                      break ;

            case 3  :
            case 6  :
            case 9  : l_bias = l_bias + l_step ;
                      l_dp = l_dp + l_step ;
                      l_cm = l_cm + l_step ;
                      CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;
                      break ;

            case 15 :
            case 16 :
            case 17 :
            case 18 :
            case 19 :
            case 20 :
            case 21 :
            case 22 :
            case 23 :
            case 24 :
            case 25 :
            case 26 :
            case 27 :
            case 28 :
            case 29 : CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf ;
                      break ;

            default : CAIRO_ERROR_MESSAGE("Unknown status %li for VP1,IBIAS\n",PROCEDURE_STATUS) ;
                      break ;
          }

        END_ON_PROCEDURE_STATUS
      }

    CAIRO_GET_PARAM("BIAS","GDS",gds_bias) ;

    CAIRO_GET_PARAM("DP","CGD",cgd_dp) ;
    CAIRO_GET_PARAM("DP","CBD",cbd_dp) ;
    CAIRO_GET_PARAM("DP","GM" ,gm_dp)  ;
    CAIRO_GET_PARAM("DP","GDS",gds_dp) ;
    CAIRO_GET_PARAM("DP","GMB",gmb_dp) ;

    CAIRO_GET_PARAM("CM","CGS",cgs_cm) ;
    CAIRO_GET_PARAM("CM","CGB",cgb_cm) ;
    CAIRO_GET_PARAM("CM","CBD",cbd_cm) ;
    CAIRO_GET_PARAM("CM","CGD",cgd_cm) ;
    CAIRO_GET_PARAM("CM","GM" ,gm_cm)  ;
    CAIRO_GET_PARAM("CM","GDS",gds_cm) ;

    CAIRO_DELETE_LOCAL_VARIABLE("vp1_called") ;

  ENDIF_NO_LOCAL_VARIABLE


  // --- Calcul de PM et UGF
```

```
  CAIRO_GET_LOCAL_VARIABLE("l_bias",l_bias) ;
  CAIRO_GET_LOCAL_VARIABLE("l_dp",l_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("l_cm",l_cm) ;

  // Calcul des parametres intermediaires
  fw0w1w2G1G2C2(gds_dp,cgd_dp,cbd_dp,
                gds_cm,gm_cm,cgs_cm,cgb_cm,cbd_cm,cgd_cm,
                cload_cal,
                w0,w1,w2,G1,G2,C2) ;

  // Calcul de la pulsation au point de transition
  wt = fwt(gm_dp,gm_cm,G1,G2,w0,w1,w2,C2) ;
  ugf_cal = fugf(wt) ;

  // Gain statique echelle lineaire
  gain_cal_lin = fgain_linear(gm_dp,gm_cm,G1,G2) ;

  // Produit gain-bande
  gbw_cal = fgbw(gain_cal_lin,w2) ;

  // Marge de phase
  pm_cal = fpm(wt,w0,w1,w2) ;

  // Frequence de coupure
  cof_cal = fcof(w2) ;

  if(loop_within == true)
    {
      goto loop_within_pm ;
    }

  if(loop_ugf == true)
    {
      goto loop_compute_ugf ;
    }


  bad_return_pm_ugf :
  CAIRO_SET_VALUE("CLOAD_OUT",cload_cal,CP_NAV) ;
  CAIRO_SET_VALUE("SR",sr_cal,CP_NAV) ;
  CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_NAV) ;
  return ;

END_PROCEDURE

// ----------------------------------------
// ------------- UGF avec minimisation de PM
// ----------------------------------------

// La procedure UGF appelle en interne la procedure VP1 pour le calcul des parametres
// petits signaux (point de polarisation).
// Les valeurs des L sont passees par des variables locales, la valeur de IBIAS par
// parametre. Les capacites sont passees par variables locales.

CAIRO_BEGIN_PROCEDURE("UGF_MIN_PM")

  // Capacite de charge
  CAIRO_TRY_GET_VALUE("CLOAD",cload_in)
    // Rappeler les parasites de routage
    // et en tenir compte
    cload_cal = cload_in + cpar ;
  IF_NO_VALUE
    CAIRO_ERROR_MESSAGE("Missing input CLOAD in %s in UGF",name) ;
  ENDIF_NO_VALUE

  // ------------------ Contraintes

  // Contrainte sur UGF
  EXAMINE_CONSTRAINT("UGF")
    CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("UGF",ugf_in) ;
                              ugf_in_mhz = ceil(ugf_in / 1.0e6) ;
                              break ;
    CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on UGF in procedure UGF of %s\n",LOCATION.c_str()) ;
                              break ;
    CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for UGF in procedure UGF of %s\n",LOCATION.c_str()) ;
                              break ;
  END_EXAMINE

  // Contrainte sur PM
  EXAMINE_CONSTRAINT("PM")
     CASE_ABOVE             : GET_CONSTRAINT_ASSOCIATED_VALUES("PM",pm_in_low) ;
                              pm_in_high = 180 ;
                              break ;
     CASE_WITHIN            : GET_CONSTRAINT_ASSOCIATED_VALUES("PM",pm_in_low,pm_in_high) ;
                              constraint_pm_within = true ;
                              break ;
    CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on PM in procedure UGF of %s\n",LOCATION.c_str()) ;
```

```
                          break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for PM in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Contrainte sur CLOAD_OUT
EXAMINE_CONSTRAINT("CLOAD_OUT")
  CASE_BELOW              : GET_CONSTRAINT_ASSOCIATED_VALUES("CLOAD_OUT",cload_max) ;
                            break ;
  CASE_NONE               : CAIRO_ERROR_MESSAGE("No constraint on CLOAD_OUT in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for CLOAD_OUT in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Objectif additionnel
objective_minimize_pm = true ;


// On rappel VP1 pour obtenir les parametres petits signaux
// C'est aussi necessaire pour la boucle de stabilite (sinon VP1 n'est jamais
// rappele car ugf est satisfait, l n'est pas modifie)

CAIRO_TRY_GET_LOCAL_VARIABLE("vp1_called",vp1_called)
IF_NO_LOCAL_VARIABLE

  RESET_PROCEDURE_STATUS ;
  CAIRO_COMPUTE("VP1") ;
  CAIRO_DELETE_LOCAL_VARIABLE("vp1_called") ;

ENDIF_NO_LOCAL_VARIABLE

// Parametres essentiels manquants issus de la polarisation
// Passage de parametres par variables locales

CAIRO_TRY_GET_VALUE("IBIAS",ibias_cal)
IF_NO_VALUE
  CAIRO_ERROR_MESSAGE("Missing input IBIAS in %s when calling VP1",name) ;
ENDIF_NO_VALUE

CAIRO_GET_LOCAL_VARIABLE("gds_bias",gds_bias) ;

CAIRO_GET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
CAIRO_GET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;

CAIRO_GET_LOCAL_VARIABLE("gm_dp" ,gm_dp ) ;
CAIRO_GET_LOCAL_VARIABLE("gds_dp",gds_dp) ;
CAIRO_GET_LOCAL_VARIABLE("gmb_dp",gmb_dp) ;

CAIRO_GET_LOCAL_VARIABLE("cgs_cm",cgs_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cgb_cm",cgb_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cbd_cm",cbd_cm) ;
CAIRO_GET_LOCAL_VARIABLE("cgd_cm",cgd_cm) ;
CAIRO_GET_LOCAL_VARIABLE("gm_cm" ,gm_cm ) ;
CAIRO_GET_LOCAL_VARIABLE("gds_cm",gds_cm) ;

CAIRO_GET_LOCAL_VARIABLE("l_bias",l_bias) ;
CAIRO_GET_LOCAL_VARIABLE("l_dp",l_dp) ;
CAIRO_GET_LOCAL_VARIABLE("l_cm",l_cm) ;

// Initialisation: on se base sur les donnees du point de polarisation

ids_bias = sign * ibias_cal ;
ids_dp = sign * ibias_cal ;
ids_cm = - sign * (ibias_cal / 2.0) ;

// Calcul des parametres intermediaires
fw0w1w2G1G2C2(gds_dp,cgd_dp,cbd_dp,
              gds_cm,gm_cm,cgs_cm,cgb_cm,cbd_cm,cgd_cm,
              cload_cal,
              w0,w1,w2,G1,G2,C2) ;

// Calcul de la pulsation au point de transition
wt = fwt(gm_dp,gm_cm,G1,G2,w0,w1,w2,C2) ;
ugf_cal = fugf(wt) ;

// Gain statique echelle lineaire
gain_cal_lin = fgain_linear(gm_dp,gm_cm,G1,G2) ;

// Produit gain-bande
gbw_cal = fgbw(gain_cal_lin,w2) ;

// Marge de phase
pm_cal = fpm(wt,w0,w1,w2) ;

// Frequence de coupure
cof_cal = fcof(w2) ;
```

```
// Sauvegarde de l'objectif
minimize_pm  = objective_minimize_pm ;

do
  {
    // ***** Boucle sur UGF

    loop_compute_ugf_min_pm :

    loop_ugf      = false ;
    loop_within   = false ;
    loop_minimize = false ;
    pm_satisfied  = false ;
    pm_too_small  = false ;
    pm_too_large  = false ;
    error_in_pm   = false ;
    minimize_pm   = objective_minimize_pm ;

    do
      {
        // ***** Boucle sur PM

        loop_within_pm_min_pm :

        // --- Boucle pour satisfaire pm_low <= pm_cal

        if(pm_cal < pm_in_low)
          {
            loop_within = true ;
            pm_too_small = true ;
            pm_too_large  = false ;

            l_bias = l_bias - l_step ;
            l_dp = l_dp - l_step ;
            l_cm = l_cm - l_step ;
            l_low = l_bias ;

            CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
            CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
            CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

            if(l_bias < l_min)
              {
                CAIRO_SET_RETURN_PROCEDURE(12) ;
                CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_SMALL) ;
                CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_LARGE) ;
                CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                goto bad_return_pm_ugf_min_pm ;
              }

            goto compute_pm_ugf_min_pm ;
          }
        else
          {
            if((constraint_pm_within == true) && (pm_in_high < pm_cal))
              {
                // On a pm_high <= pm_cal

                loop_within = true ;
                pm_too_small = false ;
                pm_too_large = true ;

                l_bias = l_bias + l_step ;
                l_dp = l_dp + l_step ;
                l_cm = l_cm + l_step ;
                l_high = l_bias ;

                CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

                goto compute_pm_ugf_min_pm ;
              }
            else
              {
                // On a pm_low <= pm_cal ou pm_low <= pm_cal << pm_high

                loop_within = false ;
                pm_too_small = false ;
                pm_too_large = false ;

                // Si l'on ne doit pas minimiser pm, alors pm est satisfait

                if(minimize_pm == false)
                  {
```

```
            pm_satisfied = true ;
          }
      }
  }

// --- Boucle pour minimiser pm_cal
// On ne passe dans cette boucle que si pm est dans l'intervalle demande

loop_minimize_pm :

if(minimize_pm == true)
  {
    if(error_in_pm == true)
      {
        loop_minimize = true ;
        // La minimisation a trop fait augmenter L. Le compute_pm_ugf a rectifie de lui-meme
        // et le signal
        minimize_pm = false ;
        pm_satisfied = true ;
      }
    else
      {
        if(pm_cal < pm_in_low)
          {
            // La minimisation est allee trop bas, il faut remonter
            loop_minimize = true ;
            minimize_pm = false ;
            pm_satisfied = true ;

            l_bias = l_bias - l_step ;
            l_dp = l_dp - l_step ;
            l_cm = l_cm - l_step ;

            CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
            CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
            CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

            goto compute_pm_ugf_min_pm ;
          }
        else
          {
            if(l_bias < l_high)
              {
                // On continue a faire diminuer PM tant que le L maximum n'est atteint
                loop_minimize = true ;

                l_bias = l_bias + l_step ;
                l_dp = l_dp + l_step ;
                l_cm = l_cm + l_step ;

                CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

                goto compute_pm_ugf_min_pm ;
              }
            else
              {
                loop_minimize = false ;
                minimize_pm = false ;
                pm_satisfied = true ;
              }
          }
      }
  }
else
  {
    loop_minimize = false ;
  }

}while(pm_satisfied == false) ;


// --- Boucle pour satisfaire ugf

// Inutile d'avoir une precision trop grande, ce qui peut poser des problemes
// de convergence
ugf_cal_mhz = ugf_cal / 1.0e6 ;

delta_ugf = fabs((ugf_in - ugf_cal) / ugf_cal) ;

if((ugf_cal_mhz < ugf_in_mhz) || (delta_ugf > epsilon2))
  {
    // Aucun contrainte n'est satisfaite (ni ugf >= ugf_in, ni ugf = ugf_in, moyennant epsilon2)

    loop_ugf = true ;
    l_bias = l_min ;
    l_dp = l_min ;
```

```
        l_cm = l_min ;

        CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
        CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
        CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

        ibias_cal = ibias_cal * (ugf_in / ugf_cal) ; // ugf_in / ugf_cal > 1

        CAIRO_SET_PARAM("IBIAS",ibias_cal) ;

        goto compute_pm_ugf_min_pm ;
      }
    else
      {
        // on a ugf >= ugf_in, on arrete
        loop_ugf = false ;
        ugf_satisfied = true ;
      }
  }while(ugf_satisfied == false) ;

// Boucle sur le SR (contrainte additionnelle)

sr_cal = fsr(ibias_cal,C2) ;

EXAMINE_CONSTRAINT("SR")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("SR",sr_in) ;

                            if(sr_cal >= sr_in)
                              {
                                CAIRO_SET_VALUE("SR",sr_cal,CP_VALID) ;
                              }
                            else
                              {
                                CAIRO_SET_RETURN_PROCEDURE(13) ;
                                CAIRO_SET_VALUE("SR",sr_cal,CP_TOO_SMALL) ;
                              }
                            break ;
  CASE_NONE               : CAIRO_SET_VALUE("SR",sr_cal,CP_VALID) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for SR in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// Boucle sur le GAIN (contrainte additionnelle)

gain_cal_db = fgain_db(gm_dp,gm_cm,G1,G2) ;

EXAMINE_CONSTRAINT("GAIN")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("GAIN",gain_in) ;

                            if(gain_cal_db >= gain_in)
                              {
                                CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_VALID) ;
                              }
                            else
                              {
                                CAIRO_SET_RETURN_PROCEDURE(14) ;
                                CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_TOO_SMALL) ;
                              }
                            break ;
  CASE_NONE               : CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_VALID) ;
                            break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for GAIN in procedure UGF of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

CAIRO_SET_VALUE("UGF",ugf_cal,CP_VALID) ;
CAIRO_SET_VALUE("PM",pm_cal,CP_VALID) ;
CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_VALID) ;
CAIRO_SET_VALUE("CLOAD_OUT",cload_cal,CP_VALID) ;
CAIRO_SET_VALUE("COF",cof_cal,CP_VALID) ;
CAIRO_SET_VALUE("GBW",gbw_cal,CP_VALID) ;

// Appel des procedures completes des fils
CAIRO_COMPUTE("BIAS","direct_size") ;
CAIRO_COMPUTE("DP","direct_size") ;
CAIRO_COMPUTE("CM","direct_size") ;

CAIRO_GET_PARAM("BIAS","W",w_bias) ;
CAIRO_GET_PARAM("DP","W",w_dp) ;
CAIRO_GET_PARAM("CM","W",w_cm) ;

CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;

return ;


// -------------------------------------------------------
```

```
// ---------------------------------- Calculs sur PM et UGF
// ------------------------------------------------------------

compute_pm_ugf_min_pm :

// --- Calcul de la polarisation

CAIRO_TRY_GET_LOCAL_VARIABLE("vp1_called",vp1_called)

  // Si la polarisation n'a pas deja ete appelee separement ou que l'on
  // se trouve deja a l'iteration > 1, on l'appelle en interne
  // Les parametres ont etes positionnes. Attention: il se peut que IBIAS
  // doivent etre repositionne (l'iteration precedente a peut-etre echoue.
  // Dans ce cas IBIAS garde le statut echec)

  RESET_PROCEDURE_STATUS ;

  while(PROCEDURE_STATUS != CP_OK)
    {
      CAIRO_COMPUTE("VP1") ;

      ON_PROCEDURE_STATUS

        switch(PROCEDURE_STATUS)
          {
            case 1  :
            case 4  :
            case 7  : if((pm_too_small == false) && (pm_too_large == false) && (loop_ugf == false))
                        {
                          // On est dans l'intervalle de PM, mais on a trop fait augmenter L (pour
                          // minimiser PM). On a trouve une borne superieure pour L qui reduit
                          // l'intervalle de PM. Il faut revenir sur ses pas

                          error_in_pm = true ;

                          l_high = l_bias ;
                          l_bias = l_bias - l_step ;
                          l_dp = l_dp - l_step ;
                          l_cm = l_cm - l_step ;

                          CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                          CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                          CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;

                          CAIRO_SET_PARAM("IBIAS",ibias_cal) ;

                          // Utilisation de la boucle pour aller a l'etiquette compute_pm_ugf_min_pm ;
                        }
                      else
                        {
                          CAIRO_SET_RETURN_PROCEDURE(10) ;
                          CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_LARGE) ;
                          CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_SMALL) ;
                          CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_LARGE) ;
                          CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                          CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                          goto bad_return_pm_ugf_min_pm ;
                        }

                      break ;

            case 2  :
            case 5  :
            case 8  : CAIRO_SET_RETURN_PROCEDURE(11) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_TOO_LARGE) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_TOO_SMALL) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf_min_pm ;
                      break ;

            case 3  :
            case 6  :
            case 9  : l_bias = l_bias + l_step ;
                      l_dp = l_dp + l_step ;
                      l_cm = l_cm + l_step ;
                      CAIRO_SET_LOCAL_VARIABLE("l_bias",l_bias) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_dp",l_dp) ;
                      CAIRO_SET_LOCAL_VARIABLE("l_cm",l_cm) ;
                      break ;

            case 15 :
            case 16 :
            case 17 :
            case 18 :
```

```
            case 19 :
            case 20 :
            case 21 :
            case 22 :
            case 23 :
            case 24 :
            case 25 :
            case 26 :
            case 27 :
            case 28 :
            case 29 : CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;
                      CAIRO_SET_VALUE("IBIAS",ibias_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("PM",pm_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("UGF",ugf_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("COF",cof_cal,CP_NAV) ;
                      CAIRO_SET_VALUE("GBW",gbw_cal,CP_NAV) ;

                      goto bad_return_pm_ugf_min_pm ;
                      break ;

            default : CAIRO_ERROR_MESSAGE("Unknown status %li for VP1,IBIAS\n",PROCEDURE_STATUS) ;
                      break ;
          }

      END_ON_PROCEDURE_STATUS
    }

  CAIRO_GET_PARAM("BIAS","GDS",gds_bias) ;

  CAIRO_GET_PARAM("DP","CGD",cgd_dp) ;
  CAIRO_GET_PARAM("DP","CBD",cbd_dp) ;
  CAIRO_GET_PARAM("DP","GM" ,gm_dp)  ;
  CAIRO_GET_PARAM("DP","GDS",gds_dp) ;
  CAIRO_GET_PARAM("DP","GMB",gmb_dp) ;

  CAIRO_GET_PARAM("CM","CGS",cgs_cm) ;
  CAIRO_GET_PARAM("CM","CGB",cgb_cm) ;
  CAIRO_GET_PARAM("CM","CBD",cbd_cm) ;
  CAIRO_GET_PARAM("CM","CGD",cgd_cm) ;
  CAIRO_GET_PARAM("CM","GM" ,gm_cm)  ;
  CAIRO_GET_PARAM("CM","GDS",gds_cm) ;

IF_NO_LOCAL_VARIABLE
  // La polarisation a deja ete appelee separement, on se contente de rappeler les
  // variables memorisees

  CAIRO_GET_LOCAL_VARIABLE("gds_bias",gds_bias) ;

  CAIRO_GET_LOCAL_VARIABLE("cgd_dp",cgd_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("cbd_dp",cbd_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("gm_dp" ,gm_dp ) ;
  CAIRO_GET_LOCAL_VARIABLE("gds_dp",gds_dp) ;
  CAIRO_GET_LOCAL_VARIABLE("gmb_dp",gmb_dp) ;

  CAIRO_GET_LOCAL_VARIABLE("cgs_cm",cgs_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cgb_cm",cgb_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cbd_cm",cbd_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("cgd_cm",cgd_cm) ;
  CAIRO_GET_LOCAL_VARIABLE("gm_cm" ,gm_cm ) ;
  CAIRO_GET_LOCAL_VARIABLE("gds_cm",gds_cm) ;

  CAIRO_DELETE_LOCAL_VARIABLE("vp1_called") ;

ENDIF_NO_LOCAL_VARIABLE


// --- Calcul de PM et UGF

CAIRO_GET_LOCAL_VARIABLE("l_bias",l_bias) ;
CAIRO_GET_LOCAL_VARIABLE("l_dp",l_dp) ;
CAIRO_GET_LOCAL_VARIABLE("l_cm",l_cm) ;

// Calcul des parametres intermediaires
fw0w1w2G1G2C2(gds_dp,cgd_dp,cbd_dp,
              gds_cm,gm_cm,cgs_cm,cgb_cm,cbd_cm,cgd_cm,
              cload_cal,
              w0,w1,w2,G1,G2,C2) ;

// Calcul de la pulsation au point de transition
wt = fwt(gm_dp,gm_cm,G1,G2,w0,w1,w2,C2) ;
ugf_cal = fugf(wt) ;

// Gain statique echelle lineaire
gain_cal_lin = fgain_linear(gm_dp,gm_cm,G1,G2) ;

// Produit gain-bande
gbw_cal = fgbw(gain_cal_lin,w2) ;
```

```
    // Marge de phase
    pm_cal = fpm(wt,w0,w1,w2) ;

    // Frequence de coupure
    cof_cal = fcof(w2) ;

    if(loop_within == true)
      {
        goto loop_within_pm_min_pm ;
      }

    if(loop_minimize == true)
      {
        goto loop_minimize_pm ;
      }

    if(loop_ugf == true)
      {
        goto loop_compute_ugf_min_pm ;
      }


    bad_return_pm_ugf_min_pm :
    CAIRO_SET_VALUE("CLOAD_OUT",cload_cal,CP_NAV) ;
    CAIRO_SET_VALUE("SR",sr_cal,CP_NAV) ;
    CAIRO_SET_VALUE("GAIN",gain_cal_db,CP_NAV) ;
    return ;

  END_PROCEDURE


CAIRO_END_DSES(OTA);


// ------------------------------------------------------------------------------------------------
// ------------------------------------------- LAYOUT -------------------------------------------
// ------------------------------------------------------------------------------------------------

CAIRO_BEGIN_LAYOUT(OTA,char *name)

  char   type    = TRANSN ;

  double height       = 0.0 ;
  double width        = 0.0 ;
  double y_vout       = 0.0 ;
  double y_vin1       = 0.0 ;
  double y_vin2       = 0.0 ;
  double ym_va        = 0.0 ;
  double ym_vb        = 0.0 ;


  CAIRO_GET_LOCAL_VARIABLE("type",type) ;

  height = CAIRO_HEIGHT(name) ;
  width = CAIRO_WIDTH(name) ;

  if(IsTransN(type))
    {
      // -----------
      // Connecteurs
      // -----------

      // VDD
      CAIRO_PHCON(ALU1,SW_ALU1,"VDD",NORTH,CAIRO_GET_COORDINATE_X("CM","source",1),height) ;

      // VSS
      CAIRO_PHCON(ALU2,SW_ALU2,"VSS",EAST,width,CAIRO_GET_COORDINATE_Y("BIAS","source",1)) ;

      // VP1
      CAIRO_PHCON(ALU2,SW_ALU2,"VP1",WEST,0.0,CAIRO_GET_COORDINATE_Y("BIAS","gate",0)) ;

      // VOUT
      y_vout = CAIRO_GET_COORDINATE_Y("DP","ab_northeast") + 2 * PITCH ;
      CAIRO_PHCON(ALU1,SW_ALU1,"VOUT",EAST,width,y_vout) ;

      // VIN1
      y_vin1 = CAIRO_GET_COORDINATE_Y("DP","ab_northwest") + PITCH ;
      CAIRO_PHCON(ALU1,SW_ALU1,"VIN1",EAST,0.0,y_vin1) ;

      // VIN1
      y_vin2 = CAIRO_GET_COORDINATE_Y("DP","ab_northwest") + 2 * PITCH ;
      CAIRO_PHCON(ALU1,SW_ALU1,"VIN2",EAST,0.0,y_vin2) ;

      // ---------------
      // Fils de routage
      // ---------------

      // VDD
```

```
    CAIRO_WIRING1("VDD",ALU1,SW_ALU1,"CM","source",1,name,"VDD",0,VERTICAL) ;

    // VSS
    CAIRO_WIRING1("VSS",ALU2,SW_ALU2,"BIAS","source",1,name,"VSS",0,HORIZONTAL) ;

    // VP1
    CAIRO_WIRING1("VP1",ALU2,SW_ALU2,"BIAS","gate",0,name,"VP1",0,HORIZONTAL) ;

    // VOUT
    CAIRO_WIRING2("VOUT",ALU1,ALU1,SW_ALU1,SW_ALU1,"CM","drain2",0,name,"VOUT",0,VERTICAL) ;

    CAIRO_WIRING3("VOUT",ALU1,ALU1,ALU1,SW_ALU1,SW_ALU1,SW_ALU1,
                  "DP","drain2",1,
                  "CM","drain2",0,y_vout,VERTICAL) ;

    // VB
    ym_vb = CAIRO_GET_COORDINATE_Y("DP","ab_northwest") + PITCH ;
    CAIRO_WIRING3("VB",ALU2,ALU2,ALU2,SW_ALU2,SW_ALU2,SW_ALU2,
                  "DP","drain1",1,
                  "CM","drain1",0,ym_vb,VERTICAL) ;

    // VIN2
    CAIRO_WIRING2("VIN2",ALU1,ALU1,SW_ALU1,SW_ALU1,"DP","gate2",1,name,"VIN2",0,VERTICAL) ;

    // VIN1
    CAIRO_WIRING2("VIN1",ALU1,ALU1,SW_ALU1,SW_ALU1,"DP","gate1",1,name,"VIN1",0,VERTICAL) ;

    // VA
    ym_va = (CAIRO_GET_COORDINATE_Y("BIAS","ab_southwest")
             + CAIRO_GET_COORDINATE_Y("DP","ab_southwest")) / 2.0 ;

    CAIRO_WIRING4("VA",ALU1,ALU1,ALU1,ALU1,SW_ALU1,SW_ALU1,SW_ALU1,SW_ALU1,
                  "BIAS","drain",0,
                  "DP","source",0,PITCH,ym_va,HORIZONTAL) ;
  }
else
  {
    // -----------
    // Connecteurs
    // -----------

    // VDD
    CAIRO_PHCON(ALU2,SW_ALU2,"VDD",EAST,width,CAIRO_GET_COORDINATE_Y("BIAS","source",1)) ;

    // VSS
    CAIRO_PHCON(ALU1,SW_ALU1,"VSS",SOUTH,CAIRO_GET_COORDINATE_X("CM","source",0),0.0) ;

    // VP1
    CAIRO_PHCON(ALU2,SW_ALU2,"VP1",WEST,0.0,CAIRO_GET_COORDINATE_Y("BIAS","gate",0)) ;

    // VOUT
    y_vout = CAIRO_GET_COORDINATE_Y("CM","ab_northeast") + PITCH ;
    CAIRO_PHCON(ALU1,SW_ALU1,"VOUT",EAST,width,y_vout) ;

    // VIN1
    y_vin1 = CAIRO_GET_COORDINATE_Y("CM","ab_northwest") + PITCH ;
    CAIRO_PHCON(ALU1,SW_ALU1,"VIN1",WEST,0.0,y_vin1) ;

    // VIN1
    y_vin2 = CAIRO_GET_COORDINATE_Y("CM","ab_northwest") + 2 * PITCH ;
    CAIRO_PHCON(ALU1,SW_ALU1,"VIN2",WEST,0.0,y_vin2) ;

    // ---------------
    // Fils de routage
    // ---------------

    // VP1
    CAIRO_WIRING1("VP1",ALU2,SW_ALU2,"BIAS","gate",0,name,"VP1",0,HORIZONTAL) ;

    // VDD
    CAIRO_WIRING1("VDD",ALU2,SW_ALU2,"BIAS","source",1,name,"VDD",0,HORIZONTAL) ;

    // VA
    ym_va = (CAIRO_GET_COORDINATE_Y("BIAS","ab_southwest")
             + CAIRO_GET_COORDINATE_Y("DP","ab_northwest")) / 2.0 ;

    CAIRO_WIRING4("VA",ALU2,ALU2,ALU1,ALU1,SW_ALU2,SW_ALU2,SW_ALU1,SW_ALU1,
                  "BIAS","drain",0,
                  "DP","source",1,PITCH,ym_va,HORIZONTAL) ;

    // VOUT
    CAIRO_WIRING3("VOUT",ALU1,ALU1,ALU1,SW_ALU1,SW_ALU1,SW_ALU1,
                  "DP","drain1",0,
                  "CM","drain2",1,y_vout,VERTICAL) ;

    CAIRO_WIRING2("VOUT",ALU1,ALU1,SW_ALU1,SW_ALU1,"CM","drain2",1,name,"VOUT",0,VERTICAL) ;

    // VB
```

```
      CAIRO_WIRING3("VB",ALU2,ALU2,ALU2,SW_ALU2,SW_ALU2,SW_ALU2,
                    "DP","drain2",0,
                    "CM","drain1",1,y_vout + PITCH,VERTICAL) ;

      // VIN2
      CAIRO_WIRING2("VIN2",ALU1,ALU1,SW_ALU1,SW_ALU1,"DP","gate1",0,name,"VIN2",0,VERTICAL) ;

      // VIN1
      CAIRO_WIRING2("VIN1",ALU1,ALU1,SW_ALU1,SW_ALU1,"DP","gate2",0,name,"VIN1",0,VERTICAL) ;

      // VSS
      CAIRO_WIRING1("VSS",ALU1,SW_ALU1,"CM","source",0,name,"VSS",0,VERTICAL) ;
    }

CAIRO_END_LAYOUT(OTA)
```

Annexe E

Code source du fichier de tests de l'OTA en $0,13\ \mu m$

```
/*************************************************************************/
/*                        Fichier : testOTA.cpp                          */
/*                                                                       */
/* Description : fichier de test pour l'OTA simple non differentiel      */
/*                                                                       */
/* Langage : C++                Version : 1.0             Date : 28/12/2004 */
/* Auteur  : Pierre Nguyen Tuong                Qualif: Pierre Nguyen Tuong */
/* License : QPL                                                         */
/*                                                                       */
/* Historique                                                            */
/* 1.0   - 28/12/2004 : version initiale                                 */
/*                                                                       */
/* Fonctions :                                                           */
/*                                                                       */
/*************************************************************************/

// Valable uniquement pour HCMOS9 !!!

#include "cairoplus.h"

#define OTA_P

int main(int argc, char *argv[])
{
  double temp      = 300.15     ;

  double vin       = 0.60       ;
  double vout      = 0.60       ;
  double ibias     = 500.0e-6   ; // 120.0e-6
  double ibias_cal = 0.0        ;
  double vp1       = 0.0        ;

  double cload_in  = 5.0e-12    ;
  double cload_max = 5.0e-12    ;
  double cload_out = 0.0        ;
  double ugf       = 55.0e+06   ;
  double ugf_cal   = 0.0        ;
  double pm_low    = 73.0       ;
  double pm_high   = 76.0       ;
  double pm_cal    = 0.0        ;
  double gbw_cal   = 0.0        ;
  double cof_cal   = 0.0        ;
  double sr_cal    = 0.0        ;
  double gain_cal  = 0.0        ;
  double veg_bias  = 0.10       ;
  double veg_dp    = 0.10       ;
  double l_init    = 4.2e-6     ;
  double l_step    = 0.0        ;

  CAIRO_NEW_CHIP("testOTA") ;

  l_step = CAIRO_GET_DOUBLE_TABLE("PHYSICAL_GRID") * MICRON * 10.0 ;
  //l_step = CAIRO_GET_DOUBLE_TABLE("PHYSICAL_GRID") * MICRON ;

  // --------
  // Creation
  // --------
```

```
  cout << "----- Appel de la creation\n" ;

#ifdef OTA_P
  CAIRO_CREATE("libOTA","OTA","OTA",TRANSP) ;
#else
  CAIRO_CREATE("libOTA","OTA","OTA",TRANSN) ;
#endif

  // ----
  // DSES
  // ----

  // ***** Calcul du point de polarisation

  cout << "----- Positionnement des parametres\n" ;
  CAIRO_SET_PARAM("OTA","TEMP" ,temp ) ;

  CAIRO_SET_PARAM("OTA","IBIAS"    ,ibias   ) ;
  CAIRO_SET_PARAM("OTA","VIN"      ,vin     ) ;
  CAIRO_SET_PARAM("OTA","VOUT"     ,vout    ) ;
  CAIRO_SET_PARAM("OTA","VEG_BIAS",veg_bias) ;
  CAIRO_SET_PARAM("OTA","VEG_DP"   ,veg_dp  ) ;

  //CAIRO_SET_PARAM("OTA","L_INIT"   ,l_init  ) ;

  cout << "----- Appel de VP1\n" ;

  RESET_PROCEDURE_STATUS ;

  while(PROCEDURE_STATUS != CP_OK)
    {
      CAIRO_COMPUTE("OTA","VP1") ;

      ON_PROCEDURE_STATUS
        switch(PROCEDURE_STATUS)
          {
            case 3  :
            case 6  :
            case 9  : l_init = l_init + l_step ;
                      CAIRO_SET_PARAM("OTA","L_INIT",l_init) ;
                      cout << "----- l_init = " << l_init << endl ;
                      break ;
            default : CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("OTA",PROCEDURE_STATUS).c_str()) ;
                      CAIRO_ERROR_MESSAGE("No loop available\n") ;
                      break ;
          }

      END_ON_PROCEDURE_STATUS
    }

  cout << "----- Recuperation de vp1\n" ;
  CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;
  cout << "----- vp1 = " << vp1 << endl ;

  // ***** Calcul du UGF

  CAIRO_SET_PARAM("OTA","CLOAD",cload_in) ;

  // Contraintes pour ugf et pm
  CAIRO_SET_PARAM_CONSTRAINT("OTA","UGF",CP_ABOVE,ugf) ;
  //CAIRO_SET_PARAM_CONSTRAINT("OTA","UGF",CP_EQUAL,ugf) ;
  //CAIRO_SET_PARAM_CONSTRAINT("OTA","PM",CP_ABOVE,pm_low) ;
  CAIRO_SET_PARAM_CONSTRAINT("OTA","PM",CP_WITHIN,pm_low,pm_high) ;
  CAIRO_SET_PARAM_CONSTRAINT("OTA","CLOAD_OUT",CP_BELOW,cload_max) ;

  //CAIRO_SET_PARAM_OBJECTIVE("OTA","PM",CP_MINIMIZE) ;

  cout << "----- Appel de UGF\n" ;

  RESET_PROCEDURE_STATUS ;

  CAIRO_COMPUTE("OTA","UGF") ;
  ON_PROCEDURE_STATUS

    CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("OTA",PROCEDURE_STATUS).c_str()) ;
    CAIRO_ERROR_MESSAGE("No loop available\n") ;

  END_ON_PROCEDURE_STATUS

  cout << "----- Recuperation de ugf\n" ;
  CAIRO_GET_PARAM("OTA","UGF",ugf_cal) ;
  cout << "----- ugf = " << ugf_cal << "  (" << ugf << ")" << endl ;

  cout << "----- Recuperation de pm\n" ;
  CAIRO_GET_PARAM("OTA","PM",pm_cal) ;
  cout << "----- pm = " << pm_cal << "  (" << pm_low << ")" << endl ;
```

```
  cout << "----- Recuperation de ibias\n" ;
  CAIRO_GET_PARAM("OTA","IBIAS",ibias_cal) ;
  cout << "----- ibias = " << ibias_cal << "  (" << ibias << ")" << endl ;

  cout << "----- Recuperation de cload_out\n" ;
  CAIRO_GET_PARAM("OTA","CLOAD_OUT",cload_out) ;
  cout << "----- cload_out = " << cload_out << "  (" << cload_in << ")" << endl ;

  cout << "----- Recuperation de gbw\n" ;
  CAIRO_GET_PARAM("OTA","GBW",gbw_cal) ;
  cout << "----- gbw = " << gbw_cal << endl ;

  cout << "----- Recuperation de cof\n" ;
  CAIRO_GET_PARAM("OTA","COF",cof_cal) ;
  cout << "----- cof = " << cof_cal << endl ;

  cout << "----- Recuperation de sr\n" ;
  CAIRO_GET_PARAM("OTA","SR",sr_cal) ;
  cout << "----- sr = " << sr_cal << endl ;

  cout << "----- Recuperation de gain\n" ;
  CAIRO_GET_PARAM("OTA","GAIN",gain_cal) ;
  cout << "----- gain = " << gain_cal << endl ;

  cout << "----- Recuperation de vp1\n" ;
  CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;
  cout << "----- vp1 = " << vp1 << endl ;

  cout << "----- reshape\n" ;
  //CAIRO_RESHAPE(70.0,250.0);
  CAIRO_RESHAPE();

  cout << "----- stabilite\n" ;
  CAIRO_STABILITY() ;

  cout << "----- Recuperation de ugf\n" ;
  CAIRO_GET_PARAM("OTA","UGF",ugf_cal) ;
  cout << "----- ugf = " << ugf_cal << "  (" << ugf << ")" << endl ;

  cout << "----- Recuperation de pm\n" ;
  CAIRO_GET_PARAM("OTA","PM",pm_cal) ;
  cout << "----- pm = " << pm_cal << "  (" << pm_low << ")" << endl ;

  cout << "----- Recuperation de ibias\n" ;
  CAIRO_GET_PARAM("OTA","IBIAS",ibias_cal) ;
  cout << "----- ibias = " << ibias_cal << "  (" << ibias << ")" << endl ;

  cout << "----- Recuperation de cload_out\n" ;
  CAIRO_GET_PARAM("OTA","CLOAD_OUT",cload_out) ;
  cout << "----- cload_out = " << cload_out << "  (" << cload_in << ")" << endl ;

  cout << "----- Recuperation de gbw\n" ;
  CAIRO_GET_PARAM("OTA","GBW",gbw_cal) ;
  cout << "----- gbw = " << gbw_cal << endl ;

  cout << "----- Recuperation de cof\n" ;
  CAIRO_GET_PARAM("OTA","COF",cof_cal) ;
  cout << "----- cof = " << cof_cal << endl ;

  cout << "----- Recuperation de sr\n" ;
  CAIRO_GET_PARAM("OTA","SR",sr_cal) ;
  cout << "----- sr = " << sr_cal << endl ;

  cout << "----- Recuperation de gain\n" ;
  CAIRO_GET_PARAM("OTA","GAIN",gain_cal) ;
  cout << "----- gain = " << gain_cal << endl ;

  cout << "----- Recuperation de vp1\n" ;
  CAIRO_GET_PARAM("OTA","VP1"  ,vp1) ;
  cout << "----- vp1 = " << vp1 << endl ;

  cout << "----- Sauvegarde\n" ;
  CAIRO_SAVE_CHIP("testOTA") ;

  exit(0) ;
}
```

Annexe F

Code source du générateur *gmc*

```
/*************************************************************************/
/*                        Fichier : gmc.cpp                              */
/*                                                                       */
/* Description : integrateur gm/c                                        */
/*                                                                       */
/* Langage : C++                Version : 1.0             Date : 08/11/2005 */
/* Auteur  : The Masked Cucumber                Qualif: The Masked Cucumber */
/*                                                                       */
/* License : QPL                                                         */
/*                                                                       */
/* Historique                                                            */
/* 1.0   - 08/11/2005 : version initiale                                 */
/*                                                                       */
/* Fonctions :                                                           */
/*                                                                       */
/*************************************************************************/

// Entrees en mode commun: vincm = vinnd = vinpd = vgsm1 =  vdd/2
// Sorties en mode commun : voutcm = voutn = voutp =  vdd/2

using namespace std ;

#include "cairoplus.h"


// ------------------------------------------------------------------------------------------------
// ------------------------------------------- CREATE -------------------------------------------
// ------------------------------------------------------------------------------------------------


CAIRO_BEGIN_CREATE(gmc,char *name)

  double surr_nwell          = SD_NWELL / 2.0 ;
  double surr_alu2           = SD_ALU2 ;
  double surr_alu3           = SD_ALU3 ;
  double max_alu2_nwell      = max(surr_nwell,surr_alu2) ;
  double max_alu2_alu3       = max(surr_alu3,surr_alu2) ;
  double max_alu3_alu2_nwell = max(max_alu2_nwell,max_alu2_alu3) ;
  double chanel_NP           = 2.0 * SD_ALU2 + 2 * SW_ALU2 ;
  double surr_chanel_NP      = max(chanel_NP / 2.0,surr_nwell) ;


  // --------------
  // Instanciations
  // --------------

  CAIRO_CREATE("libMOS","TR_MOS","m1"    ,"M1"    ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m3"    ,"M3"    ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m5"    ,"M5"    ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m7"    ,"M7"    ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m11"   ,"M11"   ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m111"  ,"M111"  ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m33"   ,"M33"   ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m333"  ,"M333"  ,TRANSN,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m55"   ,"M55"   ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m555"  ,"M555"  ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m5bias","M5bias",TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m77"   ,"M77"   ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m777"  ,"M777"  ,TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m7bias","M7bias",TRANSP,false) ;
  CAIRO_CREATE("libMOS","TR_MOS","m2"    ,"M2"    ,TRANSN,false) ;
```

```
CAIRO_CREATE("libMOS","TR_MOS","m22"   ,"M22"   ,TRANSN,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m222"  ,"M222"  ,TRANSN,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m4"    ,"M4"    ,TRANSN,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m44"   ,"M44"   ,TRANSN,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m444"  ,"M444"  ,TRANSN,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m6"    ,"M6"    ,TRANSP,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m66"   ,"M66"   ,TRANSP,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m666"  ,"M666"  ,TRANSP,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m8"    ,"M8"    ,TRANSP,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m88"   ,"M88"   ,TRANSP,false) ;
CAIRO_CREATE("libMOS","TR_MOS","m888"  ,"M888"  ,TRANSP,false) ;


// ---------------------------
// Declaration des connecteurs
// ---------------------------

CAIRO_IO("ind" ,CP_SOUTH) ;
CAIRO_IO("ipd" ,CP_SOUTH) ;
CAIRO_IO("outn",CP_EAST)  ;
CAIRO_IO("outp",CP_EAST)  ;
CAIRO_IO("bc"  ,CP_WEST)  ;
CAIRO_IO("cp"  ,CP_WEST)  ;
CAIRO_IO("bias",CP_WEST)  ;
CAIRO_IO("dd"  ,CP_NORTH) ;
CAIRO_IO("ss"  ,CP_SOUTH) ;


// -------------------------------------
// Connexions electriques et conteneurs
// -------------------------------------

// drain gate source bulk

CAIRO_CONNECT("m1"    ,"drain => d1"    ,"gate => ipd" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m3"    ,"drain => ipd"   ,"gate => bc"  ,"source => d1"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m5"    ,"drain => ipd"   ,"gate => cp"  ,"source => ips"   ,"bulk => ips"   ) ;
CAIRO_CONNECT("m7"    ,"drain => ips"   ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m11"   ,"drain => d11"   ,"gate => ipd" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m111"  ,"drain => d111"  ,"gate => ipd" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m33"   ,"drain => ind"   ,"gate => bc"  ,"source => d11"   ,"bulk => ss"    ) ;
CAIRO_CONNECT("m333"  ,"drain => outn"  ,"gate => bc"  ,"source => d111"  ,"bulk => ss"    ) ;
CAIRO_CONNECT("m55"   ,"drain => ind"   ,"gate => cp"  ,"source => d77"   ,"bulk => d77"   ) ;
CAIRO_CONNECT("m555"  ,"drain => outn"  ,"gate => cp"  ,"source => d777"  ,"bulk => d777"  ) ;
CAIRO_CONNECT("m5bias","drain => bias"  ,"gate => cp"  ,"source => d7bias","bulk => d7bias") ;
CAIRO_CONNECT("m77"   ,"drain => d77"   ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m777"  ,"drain => d777"  ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m7bias","drain => d7bias","gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m2"    ,"drain => d2"    ,"gate => ind" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m22"   ,"drain => d22"   ,"gate => ind" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m222"  ,"drain => d222"  ,"gate => ind" ,"source => ss"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m4"    ,"drain => ind"   ,"gate => bc"  ,"source => d2"    ,"bulk => ss"    ) ;
CAIRO_CONNECT("m44"   ,"drain => ipd"   ,"gate => bc"  ,"source => d22"   ,"bulk => ss"    ) ;
CAIRO_CONNECT("m444"  ,"drain => outp"  ,"gate => bc"  ,"source => d222"  ,"bulk => ss"    ) ;
CAIRO_CONNECT("m6"    ,"drain => ind"   ,"gate => cp"  ,"source => ins"   ,"bulk => ins"   ) ;
CAIRO_CONNECT("m66"   ,"drain => ipd"   ,"gate => cp"  ,"source => d88"   ,"bulk => d88"   ) ;
CAIRO_CONNECT("m666"  ,"drain => outp"  ,"gate => cp"  ,"source => d888"  ,"bulk => d888"  ) ;
CAIRO_CONNECT("m8"    ,"drain => ins"   ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m88"   ,"drain => d88"   ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;
CAIRO_CONNECT("m888"  ,"drain => d888"  ,"gate => bias","source => dd"    ,"bulk => dd"    ) ;


// -------------------------
// --------- surrounds
// --------- sur_ptrans PMOS with isolated NWELL B
// --------- sur_nwell  PMOS with NWELL B to VDD
// --------- sur_ntrans NMOS
// -------------------------

CAIRO_VERTICAL_CONTAINER("column4","m1",NOSYM,0.0           ,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m3",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m5",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu2_nwell,max_alu3_alu2_nwell,
                                   "m7",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3 ,max_alu2_alu3       ) ;

CAIRO_VERTICAL_CONTAINER("column5","m11",NOSYM,0.0           ,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m33",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m55",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu2_nwell,max_alu3_alu2_nwell,
                                   "m77",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3 ,max_alu2_alu3       ) ;

CAIRO_VERTICAL_CONTAINER("column6","m111",NOSYM,0.0           ,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m333",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3 ,max_alu2_alu3      ,
                                   "m555",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu2_nwell,max_alu3_alu2_nwell,
                                   "m777",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3 ,max_alu2_alu3       ) ;

CAIRO_VERTICAL_CONTAINER("column3","m2",NOSYM,0.0           ,0.0           ,max_alu2_alu3      ,max_alu2_alu3 ,
                                   "m4",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3      ,max_alu2_alu3 ,
                                   "m6",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu3_alu2_nwell,max_alu2_nwell,
                                   "m8",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3      ,max_alu2_alu3  ) ;
```

```
  CAIRO_VERTICAL_CONTAINER("column2","m22",NOSYM,0.0           ,0.0           ,max_alu2_alu3      ,max_alu2_alu3  ,
                                  "m44",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3      ,max_alu2_alu3  ,
                                  "m66",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu3_alu2_nwell,max_alu2_nwell,
                                  "m88",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3      ,max_alu2_alu3  ) ;

  CAIRO_VERTICAL_CONTAINER("column1","m222",NOSYM,0.0           ,0.0           ,max_alu2_alu3      ,max_alu2_alu3  ,
                                  "m444",NOSYM,surr_chanel_NP,0.0           ,max_alu2_alu3      ,max_alu2_alu3  ,
                                  "m666",SYMXY,surr_nwell    ,surr_chanel_NP,max_alu3_alu2_nwell,max_alu2_nwell,
                                  "m888",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3      ,max_alu2_alu3  ) ;

  CAIRO_VERTICAL_CONTAINER("column_bias","m5bias",SYMXY,surr_nwell,surr_chanel_NP ,max_alu3_alu2_nwell,max_alu2_nwell,
                                    "m7bias",SYMXY,max_alu2_nwell,max_alu2_nwell,max_alu2_alu3,max_alu2_alu3 ) ;

  CAIRO_HORIZONTAL_CONTAINER(name,"column_bias",NOSYM,0.0,0.0,0.0,0.0,
                                "column1"    ,NOSYM,0.0,0.0,0.0,0.0,
                                "column2"    ,NOSYM,0.0,0.0,0.0,0.0,
                                "column3"    ,NOSYM,0.0,0.0,0.0,0.0,
                                "column4"    ,NOSYM,0.0,0.0,0.0,0.0,
                                "column5"    ,NOSYM,0.0,0.0,0.0,0.0,
                                "column6"    ,NOSYM,0.0,0.0,0.0,0.0) ;

  // --------------------------
  // Declaration des parametres
  // --------------------------

  CAIRO_DECLARE_PARAM("TEMP"            ) ;

  CAIRO_DECLARE_PARAM("RATIO"           ) ;
  CAIRO_DECLARE_PARAM("RESOLUTION"      ) ;
  CAIRO_DECLARE_PARAM("BW"              ) ;
  CAIRO_DECLARE_PARAM("AINT1"           ) ;
  CAIRO_DECLARE_PARAM("OSR"             ) ;
  CAIRO_DECLARE_PARAM("ASD"             ) ;
  CAIRO_DECLARE_PARAM("VEGM3"           ) ;
  CAIRO_DECLARE_PARAM("VEGM5"           ) ;
  CAIRO_DECLARE_PARAM("LM1"             ) ;
  CAIRO_DECLARE_PARAM("LM3"             ) ;
  CAIRO_DECLARE_PARAM("LM5"             ) ;
  CAIRO_DECLARE_PARAM("LM7"             ) ;

  CAIRO_DECLARE_PARAM("UGF"             ) ;
  CAIRO_DECLARE_PARAM("SIGNAL_POWER"    ) ;
  CAIRO_DECLARE_PARAM("CAPINT"          ) ;
  CAIRO_DECLARE_PARAM("IBIAS"           ) ;
  CAIRO_DECLARE_PARAM("VBC"             ) ;
  CAIRO_DECLARE_PARAM("VCP"             ) ;
  CAIRO_DECLARE_PARAM("VBIAS"           ) ;
  CAIRO_DECLARE_PARAM("MODULATION_COEFF") ;


  // --------------------------
  // Declaration des procedures
  // --------------------------

  CAIRO_DECLARE_PROCEDURE("SIZE_GMC",CP_COMPLETE,
                          "TEMP",CP_IN,
                          "RATIO",CP_IN,"RESOLUTION",CP_IN,"BW",CP_IN,"AINT1",CP_IN,"OSR",CP_IN,"ASD",CP_IN, // SL input
                          "VEGM3",CP_IN,"VEGM5",CP_IN,"LM1",CP_IN,"LM3",CP_IN,"LM5",CP_IN,"LM7",CP_IN,       // CL input
                          "UGF",CP_OUT,"SIGNAL_POWER",CP_OUT,"CAPINT",CP_OUT,"IBIAS",CP_OUT,"VBC",CP_OUT,    // Output
                          "VCP",CP_OUT,"VBIAS",CP_OUT,"MODULATION_COEFF",CP_OUT) ;                           // Output


  // ------------------------
  // Declaration des messages
  // ------------------------

  CAIRO_DECLARE_PROCEDURE_MESSAGE(1 ,"M1: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(2 ,"M1: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(3 ,"M3: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(4 ,"M3: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(5 ,"M7: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(6 ,"M7: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(7 ,"M5: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(8 ,"M5: IBIAS is too small") ;

CAIRO_END_CREATE(gmc)


// ------------------------------------------------------------------------------------------------
// ------------------------------------------- DSES -----------------------------------------------
// ------------------------------------------------------------------------------------------------


CAIRO_BEGIN_DSES(gmc, char *name)

  double ratio      = 0.0 ;
  double resolution = 0.0 ;
```

```
double bw               = 0.0 ;
double aint1            = 0.0 ;
double osr              = 0.0 ;
double asd              = 0.0 ;
double vdd              = 0.0 ;
double vss              = 0.0 ;
double vegm3            = 0.0 ;
double vegm5            = 0.0 ;

double lm1              = 0.0 ;
double lm3              = 0.0 ;
double lm5              = 0.0 ;
double lm7              = 0.0 ;
double wm1              = 0.0 ;
double ibias            = 0.0 ;
double nibias           = 0.0 ;
double modulation_coeff = 0.0 ;
double temp             = 0.0 ;
double ugf              = 0.0 ;
double signal_power     = 0.0 ;
double capint           = 0.0 ;
double vdsm1            = 0.0 ;
double vbsm1            = 0.0 ;
double vgsm1            = 0.0 ;
double vthm1            = 0.0 ;
double vegm1            = 0.0 ;
double gmm1             = 0.0 ;
double wm3              = 0.0 ;
double vdsm3            = 0.0 ;
double vbsm3            = 0.0 ;
double vgsm3            = 0.0 ;
double vthm3            = 0.0 ;

double wm5              = 0.0 ;
double vdsm5            = 0.0 ;
double vbsm5            = 0.0 ;
double vgsm5            = 0.0 ;
double vthm5            = 0.0 ;
double wm7              = 0.0 ;
double vdsm7            = 0.0 ;
double vbsm7            = 0.0 ;
double vgsm7            = 0.0 ;
double vthm7            = 0.0 ;
int ntr                 = 15  ;
double vcp_min          = 0.0 ;
double vcp_max          = 0.0 ;
double vcp              = 0.0 ;
double vbc              = 0.0 ;
double vbias            = 0.0 ;

// Variables intermédiaires      Trouver des noms explicites !!!!!!!!!!!!!!!!
double intA             = 0.0 ;
double intB             = 0.0 ;
double a                = 0.0 ;
double b                = 0.0 ;
double c                = 0.0 ;

// Variables intermédiaires pour calcul des capa       Trouver des noms explicites !!!!!!!!!!!!!!!!
double css              = 0.0 ;
double cbc              = 0.0 ;
double ccp              = 0.0 ;
double cinns            = 0.0 ;
double cd11             = 0.0 ;
double cd2              = 0.0 ;
double cd77             = 0.0 ;
double cd22             = 0.0 ;
double cd222            = 0.0 ;
double cpar             = 0.0 ;
double cgs              = 0.0 ;
double cgb              = 0.0 ;
double cbd              = 0.0 ;
double cgd              = 0.0 ;
double csd              = 0.0 ;

double l_step           = CAIRO_GET_DOUBLE_TABLE("PHYSICAL_GRID") * MICRON ;


// -----------------------
// Parametres obligatoires
// -----------------------

// temp
CAIRO_TRY_GET_VALUE("TEMP",temp)
IF_NO_VALUE
   FATAL_ERROR_PARAM("TEMP","parameter not set",LOCATION);
ENDIF_NO_VALUE
```

```
// ----------------------------------------
// -------- Positionnement de la temperature
// ----------------------------------------

CAIRO_SET_PARAM("m1","TEMP",temp) ;
CAIRO_SET_PARAM("m3","TEMP",temp) ;
CAIRO_SET_PARAM("m5","TEMP",temp) ;
CAIRO_SET_PARAM("m7","TEMP",temp) ;


//--------------------
// -------- vdd et vss
// -------------------

CAIRO_GET_GLOBAL_VARIABLE("VDD",vdd) ;
CAIRO_GET_GLOBAL_VARIABLE("VSS",vss) ;


// ------------------------------------------------------------------
// -------------------------- SIZE_GMC ------------------------------
// ------------------------------------------------------------------

CAIRO_BEGIN_PROCEDURE("SIZE_GMC")

  //-----------------
  // -------- ratio
  // ----------------

  CAIRO_TRY_GET_VALUE("RATIO",ratio)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"RATIO: GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- RESOLUTION
  // ------------

  CAIRO_TRY_GET_VALUE("RESOLUTION",resolution)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"RESOLUTION: GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //------------
  // -------- BW
  // -----------

  CAIRO_TRY_GET_VALUE("BW",bw)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"BW: GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //---------------
  // -------- aint1
  // --------------

  CAIRO_TRY_GET_VALUE("AINT1",aint1)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"AINT1: GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- osr
  // ------------

  CAIRO_TRY_GET_VALUE("OSR",osr)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("OSR : GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- asd
  // ------------

  CAIRO_TRY_GET_VALUE("ASD",asd)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("ASD : GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- veg
  // ------------

  CAIRO_TRY_GET_VALUE("VEGM3",vegm3)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("VEGM3 : GMC specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  CAIRO_TRY_GET_VALUE("VEGM5",vegm5)
```

```
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("VEGM5 : GMC specification is not set",LOCATION) ;
ENDIF_NO_VALUE

//-----------
// -------- L
// ----------

CAIRO_TRY_GET_VALUE("LM1",lm1)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM1 : GMC specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("LM3",lm3)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM3 : GMC specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("LM5",lm5)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM5 : GMC specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("LM7",lm7)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM7 : GMC specification is not set",LOCATION) ;
ENDIF_NO_VALUE

// Initialisation des L
CAIRO_SET_PARAM("m1","L",lm1) ;
CAIRO_SET_PARAM("m3","L",lm3) ;
CAIRO_SET_PARAM("m5","L",lm5) ;
CAIRO_SET_PARAM("m7","L",lm7) ;


// --------------------------
// ---  Dimensionnement de M1
// --------------------------

// Pour obtenir une valeur approchee de vthm1 on prend
// I0 = 100 microA, vdsm1 = (vdd - vss) / 4.0, wm1 = 5 * lm1
//ibias= 100.0e-06 * ratio ; // 100.0e-06 * ratio

vdsm1 = 0.25 * vdd ;

cout << "vdsm1 = " << vdsm1 << endl ;

CAIRO_SET_PARAM("m1","VDS",vdsm1) ;
vbsm1 = 0.0 ;
CAIRO_SET_PARAM("m1","VBS",vbsm1) ;

cout << "vbsm1 = " << vbsm1 << endl ;

//  input common mode vincm = vinnd = vinpd = vgsm1 = vdd/2

vgsm1 = 0.5 * vdd ;

CAIRO_SET_PARAM("m1","VGS",vgsm1) ;

cout << "vgsm1 = " << vgsm1 << endl ;

wm1 = 5.0 * lm1 ;

CAIRO_SET_PARAM("m1","W",wm1) ;

cout << "wm1 = " << wm1 << endl ;


// La procedure rend toujours OK
cout << "m1: IDS(VGS,W,L)\n" ;
CAIRO_COMPUTE("m1","IDS(VGS,W,L)") ;

CAIRO_GET_PARAM("m1","VTH",vthm1) ;


// Coefficient de modulation
a = vgsm1 - vthm1 ;
b = vegm3 ;
c = vthm1 ;

modulation_coeff = -(2.0 * b * a+pow(b,2)-1 / (2.0 * pow(a,2) + 2.0 * b * a + pow(b,2)) * (2.0 * c * b + 2.0 * c * a + 2.0 *
                   sqrt(-pow(c,2) * pow(a,2)+4.0 * pow(a,4)+4.0 * b * pow(a,3)+2.0 * pow(b,2) * pow(a,2))) * b *
                      c-1 / (2.0 * pow(a,2) + 2.0 * b * a+pow(b,2)) * (2.0 * c * b + 2.0 * c * a + 2.0 *
                        sqrt(-pow(c,2) * pow(a,2) + 4.0 * pow(a,4) + 4.0 * b * pow(a,3) + 2.0 * pow(b,2) * pow(a,2))) *
                           a * c + pow(c,2)) / (2.0 * pow(a,2) + 2.0 * b * a + pow(b,2));

cout << "a = " << a << endl ;
cout << "b = " << b << endl ;
```

```
cout << "c = " << c << endl ;
cout << "modulation_coeff = " << modulation_coeff << endl ;


// Capacite d'integration
vegm1 = 0.5 * vdd - vthm1 ;
capint = 4 * 1.380662e-23 * temp / (vegm1 * vegm1) ;
capint = capint * (ntr / (modulation_coeff * modulation_coeff)) ;
intA = pow(10.0, resolution * 6.0 / 10.0) ; // conversion bit equivalents en réel
capint = capint * (8.0 * intA / (3.0 * (asd * asd) * osr * aint1)) ;

cout << "capaintbrut = " << capint << endl ;
cout << "modulation index = " << modulation_coeff << endl ;


// Courant de bias
intA = 2.0 * osr * bw ; // Frequence d'echantillonnage fs
intB = capint * aint1 * intA ; // Gm
ibias = 0.5 * intB * vegm1 * ratio ; // Io

cout << "ibias = " << ibias << endl  ;


// Ca vient d'ou ? Valeur codee en dur !

if(ibias <= 1.0e-6)
  {
    ibias = 1.0e-6 ;
  }

nibias = -ibias ;


CAIRO_SET_PARAM("m1","IDS",ibias) ;
CAIRO_SET_PARAM("m3","IDS",ibias) ;
CAIRO_SET_PARAM("m5","IDS",nibias) ;
CAIRO_SET_PARAM("m7","IDS",nibias) ;

cout << "ibias gmc = " << ibias << endl ;


// --------------------------
// ---  Dimensionnement de M1
// --------------------------

vdsm1 = sqrt(1 - modulation_coeff) * vegm1 + vthm1 - vegm3 ;

cout << "vdsm1 = " << vdsm1 << endl ;

CAIRO_SET_PARAM("m1","VGS",vgsm1) ;
CAIRO_SET_PARAM("m1","VDS",vdsm1) ;

cout << "vdsm1 = " << vdsm1 << endl ;


// Boucle sur wm1
RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    cout << "m1: W(VGS,L,IDS)\n" ;

    CAIRO_COMPUTE("m1","W(VGS,L,IDS)") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 4  : // IDS trop grand
                    CAIRO_SET_VALUE("IBIAS",ibias,CP_TOO_LARGE) ;
                    CAIRO_SET_RETURN_PROCEDURE(1) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 5  : // IDS trop petit
                    CAIRO_SET_VALUE("IBIAS",ibias,CP_TOO_SMALL) ;
                    CAIRO_SET_RETURN_PROCEDURE(2) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 6  : // W trop petit
                    lm1 = lm1 + l_step ;
                    cout << " lm1++ = " << lm1 << endl ;
                    CAIRO_SET_PARAM("m1","L",lm1) ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for W(VGS,L,IDS) of m1 in SIZE_GMC\n",PROCEDURE_STATUS) ;
        }
```

```
    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("m1","W",wm1) ;

cout << "after compute m1" << endl;
cout << "wm1 = " << wm1 << endl ;
cout << "lm1 = " << lm1 << endl ;


// -------------------------
// ---  Dimensionnement de M3
// -------------------------

vdsm3 = 0.5 * vdd - vdsm1 ; //vdsm1 a ete modifie

cout << "vdsm3 = " << vdsm3 << endl ;

CAIRO_SET_PARAM("m3","VDS",vdsm3) ;
vbsm3 = -vdsm1 ;

cout << "vbsm3 = " << vbsm3 << endl ;
CAIRO_SET_PARAM("m3","VBS",vbsm3) ;

cout << "vegm3 = " << vegm3 << endl ;
CAIRO_SET_PARAM("m3","VEG",vegm3) ;

cout << "ibias = " << ibias << endl ;

// Boucle sur wm3
RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    cout << "m3: W(VEG,L,IDS)\n" ;

    CAIRO_COMPUTE("m3","W(VEG,L,IDS)") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 4  : // IDS trop grand
                    CAIRO_SET_VALUE("IBIAS",ibias,CP_TOO_LARGE) ;
                    CAIRO_SET_RETURN_PROCEDURE(3) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 5  : // IDS trop petit
                    CAIRO_SET_VALUE("IBIAS",ibias,CP_TOO_SMALL) ;
                    CAIRO_SET_RETURN_PROCEDURE(4) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 6  : // W trop petit
                    lm3 = lm3 + l_step ;
                    cout << " lm3++ = " << lm3 << endl ;
                    CAIRO_SET_PARAM("m3","L",lm3) ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for W(VGS,L,IDS) of m3 in SIZE_GMC\n",PROCEDURE_STATUS) ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("m3","W",wm3) ;
CAIRO_GET_PARAM("m3","VTH",vthm3) ;
CAIRO_GET_PARAM("m3","VGS",vgsm3) ;

cout << "wm3 = " << wm3 << endl ;
cout << "vth m3 = " << vthm3 << endl;
cout << "vgs m3 = " << vgsm3 << endl;

vbc = vgsm3 + vdsm1 ;



// -------------------------
// ---  Dimensionnement de M7
// -------------------------

// Pour obtenir une valeur approchée de vthm7 on prend
// vdsm7 = (vdd + vss)/4, wm7 = 10 * lm7

vgsm7 = -(vdd + vss) * 0.5;
vbias =  vdd /2 ;
```

```
CAIRO_SET_PARAM("m7","VGS",vgsm7) ;
CAIRO_SET_PARAM("m7","VBS",vbsm7) ;

vdsm7 = -(vdd + vss) * 0.25;
CAIRO_SET_PARAM("m7","VDS",vdsm7) ;

wm7   = 10.0 * lm7;
CAIRO_SET_PARAM("m7","W",wm7) ;

cout << "m7: IDS(VGS,W,L)\n" ;
CAIRO_COMPUTE("m7","IDS(VGS,W,L)") ;
CAIRO_GET_PARAM("m7","VTH",vthm7) ;

vthm5 = vthm7; // On suppose que vbsm5 = 0 --- Cela est à discuter

// Calcul de vcp
vcp_min = sqrt(1 + modulation_coeff) * (vgsm1 - vthm1) + vthm7 + vthm1 ;
vcp_max = (vdd+vss)*0.5 + vegm5 + vthm5 - vthm7 ;
vcp     = (vcp_max + vcp_min) * 0.5 ;


vdsm7 = -(vdd + (vegm5 + vthm5) - vcp) ;

cout << "vdsm7 = " << vdsm7 << endl ;

CAIRO_SET_PARAM("m7","VDS",vdsm7) ;


// Boucle sur wm7
RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    cout << "m7: W(VGS,L,IDS)\n" ;

    CAIRO_COMPUTE("m7","W(VGS,L,IDS)") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 4  : // IDS trop grand
                    CAIRO_SET_VALUE("IBIAS",nibias,CP_TOO_LARGE) ;
                    CAIRO_SET_RETURN_PROCEDURE(5) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 5  : // IDS trop petit
                    CAIRO_SET_VALUE("IBIAS",nibias,CP_TOO_SMALL) ;
                    CAIRO_SET_RETURN_PROCEDURE(6) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 6  : // W trop petit
                    lm7 = lm7 + l_step ;
                    cout << " lm7++ = " << lm7 << endl ;
                    CAIRO_SET_PARAM("m7","L",lm7) ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for W(VGS,L,IDS) of m7 in SIZE_GMC\n",PROCEDURE_STATUS) ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("m7","W",wm7) ;

cout << "wm7 = " << wm7 << endl ;


// --------------------------
// ---  Dimensionnement de M5
// --------------------------

vdsm5 = -(vdd + vdsm7 - (vdd - vss) * 0.5);

CAIRO_SET_PARAM("m5","VDS",vdsm5) ;

// Dimensionnement de m5
// Rappel: vbsm5 = 0.0 --- A revoir

CAIRO_SET_PARAM("m5","VBS",vbsm5) ;
vgsm5 = vegm5 + vthm5;
CAIRO_SET_PARAM("m5","VGS",vgsm5) ;
```

```
// Boucle sur wm5
RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    cout << "m5: W(VGS,L,IDS)\n" ;

    CAIRO_COMPUTE("m5","W(VGS,L,IDS)") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 4  : // IDS trop grand
                    CAIRO_SET_VALUE("IBIAS",nibias,CP_TOO_LARGE) ;
                    CAIRO_SET_RETURN_PROCEDURE(7) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 5  : // IDS trop petit
                    CAIRO_SET_VALUE("IBIAS",nibias,CP_TOO_SMALL) ;
                    CAIRO_SET_RETURN_PROCEDURE(8) ;
                    goto bad_return_size_gmc ;
                    break ;

          case 6  : // W trop petit
                    lm5 = lm5 + l_step ;
                    CAIRO_SET_PARAM("m5","L",lm5) ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for W(VGS,L,IDS) of m5 in SIZE_GMC\n",PROCEDURE_STATUS) ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("m5","W",wm5) ;
cout << "wm5 = " << wm5 << endl ;

// Calcul de capint en enlevant les capas parasites
CAIRO_GET_PARAM("m1","GM",gmm1) ;
capint= gmm1 / (intA * aint1);

cout << "capaintsoft = " << capint << endl ;
cout << "gm = " << gmm1 << endl ;

// m1 == m111

CAIRO_GET_PARAM("m1","CGS",cgs) ;
CAIRO_GET_PARAM("m1","CGB",cgb) ;
CAIRO_GET_PARAM("m3","CBD",cbd) ;
css = 2.0 * cgs + cgs + 2.0 * cgb + cgb + 2.0 * cbd ;

CAIRO_GET_PARAM("m3","CGD",cgd) ;
cbc = 2.0 * cgd ;

CAIRO_GET_PARAM("m5","CGD",cgd) ;
ccp = 2.0 * cgd ;

CAIRO_GET_PARAM("m5","CSD",csd) ;
CAIRO_GET_PARAM("m5","CBD",cbd) ;
cinns = csd + cbd ;

CAIRO_GET_PARAM("m3","CSD",csd) ;
cd11 = csd ;

CAIRO_GET_PARAM("m1","CGD",cgd) ;
cd2 = cgd ;

CAIRO_GET_PARAM("m5","CSD",csd) ;
CAIRO_GET_PARAM("m5","CBD",cbd) ;
cd77 = csd + cbd ;

CAIRO_GET_PARAM("m1","CGD",cgd) ;
cd22 = cgd ;

CAIRO_GET_PARAM("m1","CGD",cgd) ;
cd222 = cgd ;

cpar = css + cbc + ccp + cinns + cd11 + cd2 + cd77 + cd22 + cd222 ;
capint = capint - cpar;

cout << "vthm1 = " << vthm1 << endl ;
cout << "vthm3 = " << vthm3 << endl ;
cout << "vthm5 = " << vthm5 << endl ;
cout << "vthm7 = " << vthm7 << endl ;

// Frequence de transition
```

```
ugf = (aint1 * bw * osr) / M_PI ;

// Puissance du signal
signal_power = 0.5 * pow(modulation_coeff,2.0) * pow(ibias,2.0) * pow(asd,2.0) ;

// -------------------------
// ---  Validation des W et L
// -------------------------

// Premiere ligne: m1 et m2

cout << "lm1 = " << lm1 << endl ;
cout << "lm3 = " << lm3 << endl ;
cout << "lm5 = " << lm5 << endl ;
cout << "lm7 = " << lm7 << endl ;

CAIRO_SET_PARAM("m1","L",lm1) ;
CAIRO_SET_PARAM("m1","W",wm1) ;
CAIRO_COMPUTE("m1","direct_size") ;

CAIRO_SET_PARAM("m11","L",lm1) ;
CAIRO_SET_PARAM("m11","W",wm1) ;
CAIRO_COMPUTE("m11","direct_size") ;

CAIRO_SET_PARAM("m111","L",lm1) ;
CAIRO_SET_PARAM("m111","W",wm1) ;
CAIRO_COMPUTE("m111","direct_size") ;

CAIRO_SET_PARAM("m2","L",lm1) ;
CAIRO_SET_PARAM("m2","W",wm1) ;
CAIRO_COMPUTE("m2","direct_size") ;

CAIRO_SET_PARAM("m22","L",lm1) ;
CAIRO_SET_PARAM("m22","W",wm1) ;
CAIRO_COMPUTE("m22","direct_size") ;

CAIRO_SET_PARAM("m222","L",lm1) ;
CAIRO_SET_PARAM("m222","W",wm1) ;
CAIRO_COMPUTE("m222","direct_size") ;

// Deuxieme ligne: m3 et m4
CAIRO_SET_PARAM("m3","L",lm3) ;
CAIRO_SET_PARAM("m3","W",wm3) ;
CAIRO_COMPUTE("m3","direct_size") ;

CAIRO_SET_PARAM("m33","L",lm3) ;
CAIRO_SET_PARAM("m33","W",wm3) ;
CAIRO_COMPUTE("m33","direct_size") ;

CAIRO_SET_PARAM("m333","L",lm3) ;
CAIRO_SET_PARAM("m333","W",wm3) ;
CAIRO_COMPUTE("m333","direct_size") ;

CAIRO_SET_PARAM("m4","L",lm3) ;
CAIRO_SET_PARAM("m4","W",wm3) ;
CAIRO_COMPUTE("m4","direct_size") ;

CAIRO_SET_PARAM("m44","L",lm3) ;
CAIRO_SET_PARAM("m44","W",wm3) ;
CAIRO_COMPUTE("m44","direct_size") ;

CAIRO_SET_PARAM("m444","L",lm3) ;
CAIRO_SET_PARAM("m444","W",wm3) ;
CAIRO_COMPUTE("m444","direct_size") ;

// Troisieme ligne
CAIRO_SET_PARAM("m5","L",lm5) ;
CAIRO_SET_PARAM("m5","W",wm5) ;
CAIRO_COMPUTE("m5","direct_size") ;

CAIRO_SET_PARAM("m55","L",lm5) ;
CAIRO_SET_PARAM("m55","W",wm5) ;
CAIRO_COMPUTE("m55","direct_size") ;

CAIRO_SET_PARAM("m5bias","L",lm5) ;
CAIRO_SET_PARAM("m5bias","W",wm5) ;
CAIRO_COMPUTE("m5bias","direct_size") ;

CAIRO_SET_PARAM("m555","L",lm5) ;
CAIRO_SET_PARAM("m555","W",wm5) ;
CAIRO_COMPUTE("m555","direct_size") ;

CAIRO_SET_PARAM("m6","L",lm5) ;
CAIRO_SET_PARAM("m6","W",wm5) ;
CAIRO_COMPUTE("m6","direct_size") ;

CAIRO_SET_PARAM("m66","L",lm5) ;
CAIRO_SET_PARAM("m66","W",wm5) ;
```

```
    CAIRO_COMPUTE("m66","direct_size") ;

    CAIRO_SET_PARAM("m666","L",lm5) ;
    CAIRO_SET_PARAM("m666","W",wm5) ;
    CAIRO_COMPUTE("m666","direct_size") ;

    // Quatrieme ligne: m7 et m8
    CAIRO_SET_PARAM("m7","L",lm7) ;
    CAIRO_SET_PARAM("m7","W",wm7) ;
    CAIRO_COMPUTE("m7","direct_size") ;

    CAIRO_SET_PARAM("m7bias","L",lm7) ;
    CAIRO_SET_PARAM("m7bias","W",wm7) ;
    CAIRO_COMPUTE("m7bias","direct_size") ;

    CAIRO_SET_PARAM("m77","L",lm7) ;
    CAIRO_SET_PARAM("m77","W",wm7) ;
    CAIRO_COMPUTE("m77","direct_size") ;

    CAIRO_SET_PARAM("m777","L",lm7) ;
    CAIRO_SET_PARAM("m777","W",wm7) ;
    CAIRO_COMPUTE("m777","direct_size") ;

    CAIRO_SET_PARAM("m8","L",lm7) ;
    CAIRO_SET_PARAM("m8","W",wm7) ;
    CAIRO_COMPUTE("m8","direct_size") ;

    CAIRO_SET_PARAM("m88","L",lm7) ;
    CAIRO_SET_PARAM("m88","W",wm7) ;
    CAIRO_COMPUTE("m88","direct_size") ;

    CAIRO_SET_PARAM("m888","L",lm7) ;
    CAIRO_SET_PARAM("m888","W",wm7) ;
    CAIRO_COMPUTE("m888","direct_size") ;


    // Parametres de retour
    CAIRO_SET_VALUE("UGF",ugf,CP_VALID) ;
    CAIRO_SET_VALUE("SIGNAL_POWER",signal_power,CP_VALID) ;
    CAIRO_SET_VALUE("CAPINT",capint,CP_VALID) ;
    CAIRO_SET_VALUE("IBIAS",ibias,CP_VALID) ;
    CAIRO_SET_VALUE("VBC",vbc,CP_VALID) ;
    CAIRO_SET_VALUE("VCP",vcp,CP_VALID) ;
    CAIRO_SET_VALUE("VBIAS",vbias, CP_VALID) ;
    CAIRO_SET_VALUE("MODULATION_COEFF",modulation_coeff,CP_VALID) ;

    // Code de retour de la procedure
    CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;

    return ;


    bad_return_size_gmc :
    CAIRO_SET_VALUE("UGF",ugf,CP_NAV) ;
    CAIRO_SET_VALUE("SIGNAL_POWER",signal_power,CP_NAV) ;
    CAIRO_SET_VALUE("CAPINT",capint,CP_NAV) ;
    CAIRO_SET_VALUE("VBC",vbc,CP_NAV) ;
    CAIRO_SET_VALUE("VCP",vcp,CP_NAV) ;
    CAIRO_SET_VALUE("VBIAS",vbias, CP_NAV) ;
    CAIRO_SET_VALUE("MODULATION_COEFF",modulation_coeff,CP_NAV) ;
    return ;

  END_PROCEDURE

  CAIRO_DEFAULT_PROCEDURE
    FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"Unknown procedure",LOCATION) ;
  END_DEFAULT_PROCEDURE

CAIRO_END_DSES(gmc)


// -------------------------------------------------------------------------------------------------
// ------------------------------------------------ LAYOUT -----------------------------------------
// -------------------------------------------------------------------------------------------------

CAIRO_BEGIN_LAYOUT(gmc,char *name)

  double height = 0.0 ;
  double width  = 0.0 ;
//double pivot  = 0.0 ;

  height = CAIRO_HEIGHT(name) ;
  width  = CAIRO_WIDTH(name)  ;

  CAIRO_PHCON(ALU2,1,"ind"  ,SOUTH,width/2 - (SD_ALU3 + SW_ALU3)/2.0 ,0.0              ) ;
  CAIRO_PHCON(ALU2,1,"ipd"  ,SOUTH,width/2 + (SD_ALU3 + SW_ALU3)/2.0 ,0.0              ) ;
  CAIRO_PHCON(ALU2,1,"outn",EAST ,width/2 - (SD_ALU3 + SW_ALU3)/2.0 ,height            ) ;
  CAIRO_PHCON(ALU2,1,"outp",EAST ,width/2 + (SD_ALU3 + SW_ALU3)/2.0 ,height            ) ;
```

```
CAIRO_PHCON(ALU2,1,"bc"  ,WEST ,0.0                              ,height / 6.0      ) ;
CAIRO_PHCON(ALU2,1,"cp"  ,WEST ,0.0                              ,2.0 * height / 6.0) ;
CAIRO_PHCON(ALU2,1,"bias",WEST ,0.0                              ,3.0 * height / 6.0) ;
CAIRO_PHCON(ALU2,1,"dd"  ,NORTH,height                           ,width / 2.0       ) ;
CAIRO_PHCON(ALU2,1,"ss"  ,SOUTH,0.0                              ,width / 2.0       ) ;


// routage des grilles sur ligne 1
// signal grille ind

#ifdef kkkkkk

CAIRO_WIRING1("ind",ALU2,SW_ALU2,"m222","gate",1,"m22","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("ind",ALU2,SW_ALU2,"m22","gate",1,"m2","gate",0,HORIZONTAL) ;

// signal grille ipd
CAIRO_WIRING1("ipd",ALU2,SW_ALU2,"m1","gate",1,"m11","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("ipd",ALU2,SW_ALU2,"m11","gate",1,"m111","gate",0,HORIZONTAL) ;

// routage des grilles sur ligne 2
// signal vbc
CAIRO_WIRING1("bc",ALU2,SW_ALU2,"m444","gate",1,"m44","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bc",ALU2,SW_ALU2,"m44","gate",1,"m4","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bc",ALU2,SW_ALU2,"m4","gate",1,"m3","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bc",ALU2,SW_ALU2,"m3","gate",1,"m33","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bc",ALU2,SW_ALU2,"m33","gate",1,"m333","gate",0,HORIZONTAL) ;

// routage des Ss sur ligne 1
// signal ss
CAIRO_WIRING1("ss",ALU2,SW_ALU2,"m222","source",1,"m22","source",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU2,SW_ALU2,"m22","source",1,"m2","source",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU2,SW_ALU2,"m2","source",1,"m1","source",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU2,SW_ALU2,"m1","source",1,"m11","source",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU2,SW_ALU2,"m11","source",1,"m111","source",0,HORIZONTAL) ;

// routage des substrats sur ligne 1
// signal B ss
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m222","bulk",1,"m22","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m22","bulk",1,"m2","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m2","bulk",1,"m1","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m1","bulk",1,"m11","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m11","bulk",1,"m111","bulk",0,HORIZONTAL) ;

// routage des substrats sur ligne 2
// signal B ss
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m444","bulk",1,"m44","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m44","bulk",1,"m4","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m4","bulk",1,"m3","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m3","bulk",1,"m33","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("ss",ALU1,SW_ALU2,"m33","bulk",1,"m333","bulk",0,HORIZONTAL) ;

// routage signal ipd
CAIRO_WIRING2("ipd",ALU3,ALU2,SW_ALU3,SW_ALU2,name,"ipd",0,"m3","drain",0,VERTICAL) ;
CAIRO_WIRING2("ipd",ALU3,ALU2,SW_ALU3,SW_ALU2,name,"ipd",0,"m1","gate",0,VERTICAL) ;

// routage signal ind
CAIRO_WIRING2("ind",ALU3,ALU2,SW_ALU3,SW_ALU2,name,"ind",0,"m4","drain",1,VERTICAL) ;
CAIRO_WIRING2("ind",ALU3,ALU2,SW_ALU3,SW_ALU2,name,"ind",0,"m2","gate",1,VERTICAL) ;

pivot  = CAIRO_GET_COORDINATE_X("m1","drain",1) + CAIRO_GET_COORDINATE_X("m3","source",1) ;

// routage des D de la ligne 1 avec les Ss de la ligne 2
CAIRO_WIRING3("d1",ALU2,ALU3,ALU2,
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m1","drain",1,
                  "m3","source",1,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m11","drain",1) + CAIRO_GET_COORDINATE_X("m33","source",1) ;

CAIRO_WIRING3("d11",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                  "m11","drain",1,
                  "m33","source",1,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m111","drain",1) + CAIRO_GET_COORDINATE_X("m333","source",1) ;

CAIRO_WIRING3("d111",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m111","drain",1,
                    "m333","source",1,
                    pivot / 2.0,HORIZONTAL) ;


pivot = CAIRO_GET_COORDINATE_X("m2","drain",0) + CAIRO_GET_COORDINATE_X("m4","source",0) ;

CAIRO_WIRING3("d2",ALU2,ALU3,ALU2,
```

```
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m2","drain",0,
                  "m4","source",0,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m22","drain",0) + CAIRO_GET_COORDINATE_X("m44","source",0) ;

CAIRO_WIRING3("d22",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                   "m22","drain",0,
                   "m44","source",0,
                   pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m222","drain",0) + CAIRO_GET_COORDINATE_X("m444","source",0) ;

CAIRO_WIRING3("d222",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m222","drain",0,
                    "m444","source",0,
                    pivot / 2.0,HORIZONTAL) ;

// routage du signal ind entre D des transistors 55 et 4

pivot = CAIRO_GET_COORDINATE_Y("m55","ab_northwest") + CAIRO_GET_COORDINATE_Y("m4","ab_northeast") ;
pivot =  CAIRO_GET_COORDINATE_Y("m55","ab_northwest") + pivot / 2.0 ;

CAIRO_WIRING3("ind",ALU3,ALU2,ALU3,
                    SW_ALU3,SW_ALU2,SW_ALU3,
                    "m55","drain",0,
                    "m4","drain",1,
                    pivot / 2.0,VERTICAL) ;

pivot = CAIRO_GET_COORDINATE_Y("m66","ab_northeast") + CAIRO_GET_COORDINATE_Y("m3","ab_northwest") ;
pivot =  CAIRO_GET_COORDINATE_Y("m3","ab_northwest") + pivot / 2.0 ;

CAIRO_WIRING3("ipd",ALU3   ,ALU2   ,ALU3,
                    SW_ALU3,SW_ALU2,SW_ALU3,
                    "m3","drain",0,
                    "m66","drain",1,
                    pivot / 2.0,VERTICAL) ;

// routage des grilles PMOS sur cp
CAIRO_WIRING1("cp",ALU2,SW_ALU2,"m666","gate",1,"m66","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("cp",ALU2,SW_ALU2,"m66","gate",1,"m6","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("cp",ALU2,SW_ALU2,"m6","gate",1,"m5","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("cp",ALU2,SW_ALU2,"m5","gate",1,"m55","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("cp",ALU2,SW_ALU2,"m55","gate",1,"m555","gate",0,HORIZONTAL) ;

// routage des grilles PMOS sur bias
CAIRO_WIRING1("bias",ALU2,SW_ALU2,"m888","gate",1,"m88","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bias",ALU2,SW_ALU2,"m88","gate",1,"m8","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bias",ALU2,SW_ALU2,"m8","gate",1,"m7","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bias",ALU2,SW_ALU2,"m7","gate",1,"m77","gate",0,HORIZONTAL) ;
CAIRO_WIRING1("bias",ALU2,SW_ALU2,"m77","gate",1,"m777","gate",0,HORIZONTAL) ;

// routage des Ss PMOS sur ligne dd
CAIRO_WIRING1("dd",ALU2,SW_ALU2,"m888","source",1,"m88","source",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU2,SW_ALU2,"m88","source",1,"m8","source",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU2,SW_ALU2,"m8","source",1,"m7","source",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU2,SW_ALU2,"m7","source",1,"m77","source",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU2,SW_ALU2,"m77","source",1,"m777","source",0,HORIZONTAL) ;

// routage des substrats PMOS 888 a 777 sur dd
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m888","bulk",1,"m88","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m88","bulk",1,"m8","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m8","bulk",1,"m7","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m7","bulk",1,"m77","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m77","bulk",1,"m777","bulk",0,HORIZONTAL) ;

// routage des substrats PMOS 666 a 555 sur dd
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m666","bulk",1,"m66","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m66","bulk",1,"m6","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m6","bulk",1,"m5","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m5","bulk",1,"m55","bulk",0,HORIZONTAL) ;
CAIRO_WIRING1("dd",ALU1,SW_ALU2,"m55","bulk",1,"m555","bulk",0,HORIZONTAL) ;

// routage des Ds de la ligne 4 avec les Ss de la ligne 3

pivot = CAIRO_GET_COORDINATE_X("m7","drain",1) + CAIRO_GET_COORDINATE_X("m5","source",1) ;

CAIRO_WIRING3("d7",ALU2,ALU3,ALU2,
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m7","drain",1,
                  "m5","source",1,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m77","drain",1) + CAIRO_GET_COORDINATE_X("m55","source",1) ;
```

```
CAIRO_WIRING3("d77",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                   "m77","drain",1,
                   "m55","source",1,
                   pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m777","drain",1) + CAIRO_GET_COORDINATE_X("m555","source",1) ;

CAIRO_WIRING3("d777",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m777","drain",1,
                    "m555","source",1,
                    pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m8","drain",0) + CAIRO_GET_COORDINATE_X("m6","source",0) ;

CAIRO_WIRING3("d8",ALU2,ALU3,ALU2,
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m8","drain",0,
                  "m6","source",0,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m88","drain",0) + CAIRO_GET_COORDINATE_X("m66","source",0) ;

CAIRO_WIRING3("d88",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                   "m88","drain",0,
                   "m66","source",0,
                   pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m888","drain",0) + CAIRO_GET_COORDINATE_X("m666","source",0) ;

CAIRO_WIRING3("d888",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m888","drain",0,
                    "m666","source",0,
                    pivot / 2.0,HORIZONTAL) ;

// routage des Ds de la ligne 3 avec les Ds de la ligne 2

pivot = CAIRO_GET_COORDINATE_X("m5","drain",1) + CAIRO_GET_COORDINATE_X("m3","drain",1) ;

CAIRO_WIRING3("d5",ALU2,ALU3,ALU2,
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m5","drain",1,
                  "m3","drain",1,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m55","drain",1) + CAIRO_GET_COORDINATE_X("m33","drain",1) ;

CAIRO_WIRING3("d55",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                   "m55","drain",1,
                   "m33","drain",1,
                   pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m555","drain",1) + CAIRO_GET_COORDINATE_X("m333","drain",1) ;

CAIRO_WIRING3("d555",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m555","drain",1,
                    "m333","drain",1,
                    pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m6","drain",0) + CAIRO_GET_COORDINATE_X("m4","drain",0) ;

CAIRO_WIRING3("d6",ALU2,ALU3,ALU2,
                  SW_ALU2,SW_ALU3,SW_ALU2,
                  "m6","drain",0,
                  "m4","drain",0,
                  pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m66","drain",0) + CAIRO_GET_COORDINATE_X("m44","drain",0) ;

CAIRO_WIRING3("d66",ALU2,ALU3,ALU2,
                   SW_ALU2,SW_ALU3,SW_ALU2,
                   "m66","drain",0,
                   "m44","drain",0,
                   pivot / 2.0,HORIZONTAL) ;

pivot = CAIRO_GET_COORDINATE_X("m666","drain",0) + CAIRO_GET_COORDINATE_X("m444","drain",0) ;

CAIRO_WIRING3("d666",ALU2,ALU3,ALU2,
                    SW_ALU2,SW_ALU3,SW_ALU2,
                    "m666","drain",0,
                    "m444","drain",0,
                    pivot / 2.0,HORIZONTAL) ;
```

```
  CAIRO_WIRING1("outp",ALU2,SW_ALU2,"m666","drain",1,"m444","drain",0,VERTICAL) ;
  CAIRO_WIRING1("outn",ALU2,SW_ALU2,"m555","drain",1,"m333","drain",0,VERTICAL) ;

#endif


CAIRO_END_LAYOUT(gmc)
```

Annexe G

Code source du fichier de tests du *gmc* en $0,13\ \mu m$

```
/*****************************************************************************/
/*                        Fichier : testGMC.cpp                              */
/*                                                                           */
/* Description : fichier de tests de l'integrateur gm/c                      */
/*                                                                           */
/* Langage : C++                Version : 1.0              Date : 08/11/2005 */
/* Auteur  : The Masked Cucumber                 Qualif: The Masked Cucumber */
/*                                                                           */
/* License : QPL                                                             */
/*                                                                           */
/* Historique                                                                */
/* 1.0   - 08/11/2005 : version initiale                                     */
/*                                                                           */
/* Fonctions :                                                               */
/*                                                                           */
/*****************************************************************************/

#include <iostream>
#include <fstream>

using namespace std ;

#include "cairoplus.h"

#define performe_stability 1

int main(int argc, char *argv[])
{
  // Specification de niveau systeme
  double ratio              = 1         ;
  double resolution         = 10.0      ;
  double bw                 = 500e03    ;
  double osr                = 64.0      ;
  double asd                = 0.5       ;

  // Caracteristique de contre-reaction DAC RZ
  double tau                = 0.75      ;

  // Integrator gain
  double aint1              = 0.0333    ;

  // Coefficient de contre-reaction differentielle
  double coeff1             = 0.5 / tau ;

  // Specifications de niveau circuit    // bon      12-500   orig
  double vdd                = 1.0       ; // 1.0      1.2      0.7
  double vss                = 0.0       ; // 0.0      0.0      0.0
  double vegm3              = 0.106     ; // 0.106    0.1      0.1
  double vegm5              = -0.1      ; // -0.1     -0.1     -0.2
  double lm1                = 3.5e-06   ; // 3.5e-06  2.0e-06  7.0e-06
  double lm3                = 2.8e-06   ; // 2.8e-06  1.0e-06  2.0e-06
  double lm5                = 2.8e-06   ; // 2.8e-06  1.0e-06  2.0e-06
  double lm7                = 3.5e-06   ; // 3.5e-06  2.0e-06  7.0e-06

  // Parametres de sortie
  double ugf                = 0.0       ;
  double signal_power       = 0.0       ;
```

```
double capint1           = 0.0       ;
double ibias1            = 0.0       ;
double vbc1              = 0.0       ;
double vcp1              = 0.0       ;
double vbias1            = 0.0       ;
double modulation_coeff1 = 0.0       ;

cout << "--- Parametres ------------" << endl ;
cout << "RATIO = "      << ratio        << endl ;
cout << "RESOLUTION = " << resolution   << endl ;
cout << "BW    = "      << bw           << endl ;
cout << "Aint1 = "      << aint1        << endl ;
cout << "coeff1= "      << coeff1       << endl ;
cout << "OSR   = "      << osr          << endl ;
cout << "VDD   = "      << vdd          << endl ;
cout << "VSS   = "      << vss          << endl ;
cout << "VEGM3 = "      << vegm3        << endl ;
cout << "VEGM5 = "      << vegm5        << endl ;
cout << "Lm1   = "      << lm1          << endl ;
cout << "Lm3   = "      << lm3          << endl ;
cout << "Lm5   = "      << lm5          << endl ;
cout << "Lm7   = "      << lm7          << endl ;
cout << "---------------------------" << endl ;


CAIRO_NEW_CHIP("test_gmc") ;

// --------
// Creation
// --------

CAIRO_CREATE("libGMC","gmc","gmc") ;

CAIRO_SET_GLOBAL_VARIABLE("VDD",vdd) ;
CAIRO_SET_GLOBAL_VARIABLE("VSS",vss) ;

// ----
// DSES
// ----

CAIRO_SET_PARAM("gmc","TEMP"      ,300.15    ) ;
CAIRO_SET_PARAM("gmc","RATIO"     ,ratio     ) ;
CAIRO_SET_PARAM("gmc","RESOLUTION",resolution) ;
CAIRO_SET_PARAM("gmc","BW"        ,bw        ) ;
CAIRO_SET_PARAM("gmc","AINT1"     ,aint1     ) ;
CAIRO_SET_PARAM("gmc","OSR"       ,osr       ) ;
CAIRO_SET_PARAM("gmc","ASD"       ,asd       ) ;
CAIRO_SET_PARAM("gmc","VEGM3"     ,vegm3     ) ;
CAIRO_SET_PARAM("gmc","VEGM5"     ,vegm5     ) ;
CAIRO_SET_PARAM("gmc","LM1"       ,lm1       ) ;
CAIRO_SET_PARAM("gmc","LM3"       ,lm3       ) ;
CAIRO_SET_PARAM("gmc","LM5"       ,lm5       ) ;
CAIRO_SET_PARAM("gmc","LM7"       ,lm7       ) ;


RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    CAIRO_COMPUTE("gmc","SIZE_GMC") ;

    ON_PROCEDURE_STATUS
      switch(PROCEDURE_STATUS)
        {
          case 1  :
          case 2  :
          case 3  :
          case 4  :
          case 5  :
          case 6  :
          case 7  :
          case 8  : CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("gmc",PROCEDURE_STATUS).c_str()) ;
                    CAIRO_ERROR_MESSAGE("No loop available\n") ;
                    break ;
          default : CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("gmc",PROCEDURE_STATUS).c_str()) ;
                    CAIRO_ERROR_MESSAGE("No loop available\n") ;
                    break ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("gmc","UGF",ugf) ;
CAIRO_GET_PARAM("gmc","SIGNAL_POWER",signal_power) ;
CAIRO_GET_PARAM("gmc","CAPINT",capint1) ;
CAIRO_GET_PARAM("gmc","IBIAS",ibias1) ;
CAIRO_GET_PARAM("gmc","VBC",vbc1) ;
CAIRO_GET_PARAM("gmc","VCP",vcp1) ;
```

```
  CAIRO_GET_PARAM("gmc","VBIAS",vbias1) ;
  CAIRO_GET_PARAM("gmc","MODULATION_COEFF",modulation_coeff1) ;

  cout << "--- Final ----------------------------------------" << endl ;
  cout << "main : UGF              = " << ugf              << endl ;
  cout << "main : SIGNAL_POWER     = " << signal_power     << endl ;
  cout << "main : CAPINT           = " << capint1          << endl ;
  cout << "main : IBIAS            = " << ibias1           << endl ;
  cout << "main : VBC              = " << vbc1             << endl ;
  cout << "main : VCP              = " << vcp1             << endl ;
  cout << "main : VBIAS            = " << vbias1           << endl ;
  cout << "main : MODULATION_COEFF = " << modulation_coeff1 << endl ;
  cout << "------------------------------------------------" << endl ;

  CAIRO_RESHAPE(20,50000.0) ;

#ifdef performe_stability
  CAIRO_STABILITY();

  CAIRO_GET_PARAM("gmc","UGF",ugf) ;
  CAIRO_GET_PARAM("gmc","SIGNAL_POWER",signal_power) ;
  CAIRO_GET_PARAM("gmc","CAPINT",capint1) ;
  CAIRO_GET_PARAM("gmc","IBIAS",ibias1) ;
  CAIRO_GET_PARAM("gmc","VBC",vbc1) ;
  CAIRO_GET_PARAM("gmc","VCP",vcp1) ;
  CAIRO_GET_PARAM("gmc","VBIAS",vbias1) ;
  CAIRO_GET_PARAM("gmc","MODULATION_COEFF",modulation_coeff1) ;

  cout << "--- Final ----------------------------------------" << endl ;
  cout << "main : UGF              = " << ugf              << endl ;
  cout << "main : SIGNAL_POWER     = " << signal_power     << endl ;
  cout << "main : CAPINT           = " << capint1          << endl ;
  cout << "main : IBIAS            = " << ibias1           << endl ;
  cout << "main : VBC              = " << vbc1             << endl ;
  cout << "main : VCP              = " << vcp1             << endl ;
  cout << "main : VBIAS            = " << vbias1           << endl ;
  cout << "main : MODULATION_COEFF = " << modulation_coeff1 << endl ;
  cout << "------------------------------------------------" << endl ;
#endif

  {
    // Generation automatique des fichiers de simulation

    ofstream cirFile ;

    // ----------
    // Analyse AC
    // ----------

    cirFile.open("simulation_ac_gmc.cir",ios::out) ;

    if(cirFile == NULL)
      {
        cerr << "Problem when opening data file simulation_ac_gmc.cir" << endl ;
        exit(1) ;
      }

    cirFile << "* 1st Integrator AC analysis *\n" ;
    cirFile << "\n" ;
    cirFile << "******************\n" ;
    cirFile << "* ALIMENTATIONS *\n" ;
    cirFile << "\n" ;
    cirFile << "ibias bias ss " << ibias1 << endl ;
    cirFile << "\n" ;
    cirFile << "vss ss 0 0.0\n" ;
    cirFile << "vdd dd ss " << vdd  << endl ;
    cirFile << "vbc bc ss " << vbc1 << endl ;
    cirFile << "vcp cp ss " << vcp1 << endl ;
    cirFile << "voutp outp ss " << vdd / 2.0 << endl ;
    cirFile << "voutn outn ss " << vdd / 2.0 << endl ;
    cirFile << "cn ind 0 " << capint1 << endl ;
    cirFile << "cp ipd 0 " << capint1 << endl ;
    cirFile << "inp dd ipd ac 1\n" ;
    cirFile << "inn dd ind ac -1\n" ;
    cirFile << "\n" ;
    cirFile << "* --------------------------------*\n" ;
    cirFile << "* Génération du circuit           *\n" ;
    cirFile << "* --------------------------------*\n" ;
    cirFile << "\n" ;
    cirFile << ".include gmc.spi\n" ;
    cirFile << "Xint ss dd bias cp bc outp outn ipd ind gmc\n" ;
    cirFile << "\n" ;
    cirFile << "* --------------------------------*\n" ;
    cirFile << "* Analysis                        *\n" ;
    cirFile << "* --------------------------------*\n" ;
    cirFile << "\n" ;
    cirFile << ".op\n" ;
    cirFile << ".ac dec 100 1.0e00 1.0e09\n" ;
```

```
cirFile << ".plot ac idb(voutp) idb(voutn)\n" ;
cirFile << ".option eps=1e-8 be hmax=0.01u captab nowavecomplex\n" ;
cirFile << "\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "* Technologie                     *\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "\n" ;
cirFile << "\n" ;
cirFile << ".PARAM MOSLL_DEV=0\n" ;
cirFile << "\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_poly.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_active.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_go1.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/mos_bsim3_LL.lib mosLL_stat\n" ;
cirFile << "\n" ;
cirFile << "\n" ;
cirFile << ".end\n" ;

cirFile.close() ;


// -------------
// Analyse bruit
// -------------

cirFile.open("simulation_bruit_gmc.cir",ios::out) ;

if(cirFile == NULL)
  {
    cerr << "Problem when opening data file simulation_bruit_gmc.cir" << endl ;
    exit(1) ;
  }

cirFile << "* 1st Integrator noise analysis *\n" ;
cirFile << "\n" ;
cirFile << "******************\n" ;
cirFile << "* ALIMENTATIONS *\n" ;
cirFile << "\n" ;
cirFile << "ibias bias ss " << ibias1 << endl ;
cirFile << "\n" ;
cirFile << "vss ss 0 0.0\n" ;
cirFile << "vdd dd ss " << vdd  << endl ;
cirFile << "vbc bc ss " << vbc1 << endl ;
cirFile << "vcp cp ss " << vcp1 << endl ;
cirFile << "voutp outp ss " << vdd / 2.0 << endl ;
cirFile << "voutn outn ss " << vdd / 2.0 << endl ;
cirFile << "cn ind 0 " << capint1 << endl ;
cirFile << "cp ipd 0 " << capint1 << endl ;
cirFile << "inp dd ipd ac 1\n" ;
cirFile << "inn dd ind ac -1\n" ;
cirFile << "\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "* Génération du circuit           *\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "\n" ;
cirFile << ".include gmc.spi\n" ;
cirFile << "Xint ss dd bias cp bc outp outn ipd ind gmc\n" ;
cirFile << "\n" ;
cirFile << "* --------------------------------*\n";
cirFile << "* Output I-V Conversion            *\n";
cirFile << "* --------------------------------*\n";
cirFile << "\n" ;
cirFile << "ha outpv 0 voutp 1\n" ;
cirFile << "hb outnv 0 voutn 1\n" ;
cirFile << "\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "* Analysis                        *\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "\n" ;
cirFile << ".op\n" ;
cirFile << ".ac dec 100 1.0e02 " << bw << endl ;
cirFile << ".noise v(outpv) inp 10\n" ;
cirFile << ".plot noise db(inoise) db(onoise)\n" ;
cirFile << ".print noise inoise\n" ;
cirFile << ".option eps=1e-8 be hmax=0.01u captab nowavecomplex\n" ;
cirFile << "\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "* Technologie                     *\n" ;
cirFile << "* --------------------------------*\n" ;
cirFile << "\n" ;
cirFile << "\n" ;
cirFile << ".PARAM MOSLL_DEV=0\n" ;
cirFile << "\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_poly.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_active.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_go1.lib PRO_stat\n" ;
cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/mos_bsim3_LL.lib mosLL_stat\n" ;
cirFile << "\n" ;
```

```
    cirFile << "\n" ;
    cirFile << ".end\n" ;

    cirFile.close() ;
  }

  CAIRO_SAVE_CHIP("test_gmc") ;

  return 0 ;
}
```

Annexe H

Code source du générateur filtre

```
/*************************************************************************/
/*                        Fichier : filter_o3.cpp                        */
/*                                                                       */
/* Description : filtre pour sigma-delta utilisant trois integrateurs gm/c   */
/*               (filtre d'ordre 3)                                      */
/*                                                                       */
/* Langage : C++                Version : 1.0              Date : 08/11/2005 */
/* Auteur  : The Masked Cucumber                Qualif: The Masked Cucumber */
/*                                                                       */
/* License : QPL                                                         */
/*                                                                       */
/* Historique                                                            */
/* 1.0   - 08/11/2005 : version initiale                                 */
/*                                                                       */
/* Fonctions :                                                           */
/*                                                                       */
/*************************************************************************/


#include "cairoplus.h"

// ---------------------------------------------------------------------------------------------------
// ---------------------------------------------- CREATE -------------------------------------------
// ---------------------------------------------------------------------------------------------------

CAIRO_BEGIN_CREATE(filter_o3,char *name)

  // --------------
  // Instanciations
  // --------------

  CAIRO_CREATE("libGMC","gmc","gmc_1") ;
  CAIRO_CREATE("libGMC","gmc","gmc_2") ;
  CAIRO_CREATE("libGMC","gmc","gmc_3") ;


  // ---------------------------
  // Declaration des connecteurs
  // ---------------------------

  CAIRO_IO("in_minus" ,CP_SOUTH) ;
  CAIRO_IO("in_plus"  ,CP_SOUTH) ;
  CAIRO_IO("out_minus",CP_EAST)  ;
  CAIRO_IO("out_plus" ,CP_EAST)  ;

  CAIRO_IO("bc"       ,CP_WEST)  ;
  CAIRO_IO("cp"       ,CP_WEST)  ;
  CAIRO_IO("bias"     ,CP_WEST)  ;

  CAIRO_IO("dd"       ,CP_NORTH) ;
  CAIRO_IO("ss"       ,CP_SOUTH) ;


  // ------------------------------------
  // Connexions electriques et conteneurs
  // ------------------------------------

  // ind ipd outn outp bc cp bias dd ss

  CAIRO_CONNECT("gmc_1","ind => in_minus","ipd => in_plus","outn => ind_2","outp => ipd_2",
                        "bc => bc","cp => cp","bias => bias","dd => dd","ss => ss") ;
```

```
CAIRO_CONNECT("gmc_2","ind => ind_2","ipd => ipd_2","outn => ind_3","outp => ipd_3",
                      "bc => bc","cp => cp","bias => bias","dd => dd","ss => ss") ;

CAIRO_CONNECT("gmc_3","ind => ind_3","ipd => ipd_3","outn => out_minus","outp => out_plus",
                      "bc => bc","cp => cp","bias => bias","dd => dd","ss => ss") ;


// -------------------------
// --------- surrounds
// --------- sur_ptrans PMOS with isolated NWELL bulk
// --------- sur_nwell  PMOS with NWELL bulk to VDD
// --------- sur_ntrans NMOS
// -------------------------

CAIRO_HORIZONTAL_CONTAINER(name,"gmc_1",NOSYM,0.0,0.0,0.0,0.0,
                               "gmc_2",NOSYM,0.0,0.0,0.0,0.0,
                               "gmc_3",NOSYM,0.0,0.0,0.0,0.0) ;


// -------------------------
// Declaration des parametres
// -------------------------

CAIRO_DECLARE_PARAM("TEMP"              ) ;

CAIRO_DECLARE_PARAM("RESOLUTION"        ) ;
CAIRO_DECLARE_PARAM("BW"                ) ;

CAIRO_DECLARE_PARAM("RATIO_1"           ) ;
CAIRO_DECLARE_PARAM("RATIO_2"           ) ;
CAIRO_DECLARE_PARAM("RATIO_3"           ) ;

CAIRO_DECLARE_PARAM("AINT1_1"           ) ;
CAIRO_DECLARE_PARAM("AINT1_2"           ) ;
CAIRO_DECLARE_PARAM("AINT1_3"           ) ;

CAIRO_DECLARE_PARAM("OSR"               ) ;
CAIRO_DECLARE_PARAM("ASD"               ) ;
CAIRO_DECLARE_PARAM("VEGM3"             ) ;
CAIRO_DECLARE_PARAM("VEGM5"             ) ;
CAIRO_DECLARE_PARAM("LM1"               ) ;
CAIRO_DECLARE_PARAM("LM3"               ) ;
CAIRO_DECLARE_PARAM("LM5"               ) ;
CAIRO_DECLARE_PARAM("LM7"               ) ;

CAIRO_DECLARE_PARAM("POWER_CONSUMPTION" ) ;

CAIRO_DECLARE_PARAM("VBC"               ) ;
CAIRO_DECLARE_PARAM("VCP"               ) ;
CAIRO_DECLARE_PARAM("VBIAS"             ) ;

CAIRO_DECLARE_PARAM("CAPINT_1"          ) ;
CAIRO_DECLARE_PARAM("IBIAS_1"           ) ;
CAIRO_DECLARE_PARAM("MODULATION_COEFF_1") ;

CAIRO_DECLARE_PARAM("CAPINT_2"          ) ;
CAIRO_DECLARE_PARAM("IBIAS_2"           ) ;
CAIRO_DECLARE_PARAM("MODULATION_COEFF_2") ;

CAIRO_DECLARE_PARAM("CAPINT_3"          ) ;
CAIRO_DECLARE_PARAM("IBIAS_3"           ) ;
CAIRO_DECLARE_PARAM("MODULATION_COEFF_3") ;


// -------------------------
// Declaration des procedures
// -------------------------

CAIRO_DECLARE_PROCEDURE("SIZE_FILTER_03",CP_COMPLETE,
                        "TEMP",CP_IN,
                        "RESOLUTION",CP_IN,"BW",CP_IN,"AINT1_1",CP_IN,                                  // SL input
                        "RATIO_1",CP_IN,"RATIO_2",CP_IN,"RATIO_3",CP_IN,                                // SL input
                        "AINT1_2",CP_IN,"AINT1_3",CP_IN,"OSR",CP_IN,"ASD",CP_IN,                        // SL input
                        "VEGM3",CP_IN,"VEGM5",CP_IN,"LM1",CP_IN,"LM3",CP_IN,"LM5",CP_IN,"LM7",CP_IN,    // CL input
                        "POWER_CONSUMPTION",CP_OUT,                                                     // Output
                        "VBC",CP_OUT,"VCP",CP_OUT,"VBIAS",CP_OUT,                                       // Output
                        "CAPINT_1",CP_OUT,"IBIAS_1",CP_OUT,                                             // Output
                        "MODULATION_COEFF_1",CP_OUT,                                                    // Output
                        "CAPINT_2",CP_OUT,"IBIAS_2",CP_OUT,                                             // Output
                        "MODULATION_COEFF_2",CP_OUT,                                                    // Output
                        "CAPINT_3",CP_OUT,"IBIAS_3",CP_OUT,                                             // Output
                        "MODULATION_COEFF_3",CP_OUT) ;                                                  // Output

// ------------------------
// Declaration des messages
// ------------------------
```

```
  CAIRO_DECLARE_PROCEDURE_MESSAGE(1  ,"GMC_1: M1: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(2  ,"GMC_1: M1: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(3  ,"GMC_1: M3: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(4  ,"GMC_1: M3: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(5  ,"GMC_1: M7: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(6  ,"GMC_1: M7: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(7  ,"GMC_1: M5: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(8  ,"GMC_1: M5: IBIAS is too small") ;

  CAIRO_DECLARE_PROCEDURE_MESSAGE(9  ,"GMC_2: M1: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(10 ,"GMC_2: M1: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(11 ,"GMC_2: M3: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(12 ,"GMC_2: M3: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(13 ,"GMC_2: M7: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(14 ,"GMC_2: M7: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(15 ,"GMC_2: M5: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(16 ,"GMC_2: M5: IBIAS is too small") ;

  CAIRO_DECLARE_PROCEDURE_MESSAGE(17 ,"GMC_3: M1: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(18 ,"GMC_3: M1: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(19 ,"GMC_3: M3: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(20 ,"GMC_3: M3: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(21 ,"GMC_3: M7: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(22 ,"GMC_3: M7: IBIAS is too small") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(23 ,"GMC_3: M5: IBIAS is too large") ;
  CAIRO_DECLARE_PROCEDURE_MESSAGE(24 ,"GMC_3: M5: IBIAS is too small") ;

  CAIRO_DECLARE_PROCEDURE_MESSAGE(25 ,"POWER_CONSUMPTION is too large") ;

CAIRO_END_CREATE(filter_o3)


// ------------------------------------------------------------------------------------------------------
// ------------------------------------------- DSES -----------------------------------------------------
// ------------------------------------------------------------------------------------------------------

//  input common mode vincm = vinnd = vinpd = vgsm1 =  vdd/2
//  output common mode voutcm = voutn = voutp =  vdd/2

CAIRO_BEGIN_DSES(filter_o3, char *name)

  double resolution         = 0.0   ;
  double bw                 = 0.0   ;
  double ratio_1            = 0.0   ;
  double ratio_2            = 0.0   ;
  double ratio_3            = 0.0   ;
  double aint1_1            = 0.0   ;
  double aint1_2            = 0.0   ;
  double aint1_3            = 0.0   ;
  double osr                = 0.0   ;
  double asd                = 0.0   ;
  double vdd                = 0.0   ;
  double vss                = 0.0   ;
  double vegm3              = 0.0   ;
  double vegm5              = 0.0   ;

  double lm1                = 0.0   ;
  double lm3                = 0.0   ;
  double lm5                = 0.0   ;
  double lm7                = 0.0   ;

  double power_consumption  = 0.0   ;

  double capint_1           = 0.0   ;
  double ibias_1            = 0.0   ;
  double vbc_1              = 0.0   ;
  double vcp_1              = 0.0   ;
  double vbias_1            = 0.0   ;
  double modulation_coeff_1 = 0.0   ;

  double capint_2           = 0.0   ;
  double ibias_2            = 0.0   ;
  double vbc_2              = 0.0   ;
  double vcp_2              = 0.0   ;
  double vbias_2            = 0.0   ;
  double modulation_coeff_2 = 0.0   ;

  double capint_3           = 0.0   ;
  double ibias_3            = 0.0   ;
  double vbc_3              = 0.0   ;
  double vcp_3              = 0.0   ;
  double vbias_3            = 0.0   ;
  double modulation_coeff_3 = 0.0   ;

  const double ratio_step   = 0.1   ;
  const double min_ratio    = 0.1   ;
  double current_ratio1     = 1.0   ;
  double current_ratio2     = 1.0   ;
```

```
double current_ratio3     = 1.0   ;
double max_power          = 0.0   ;
bool   power_constraint   = false ;

double temp               = 0.0   ;



// -----------------------
// Parametres obligatoires
// -----------------------

// temp
CAIRO_TRY_GET_VALUE("TEMP",temp)
IF_NO_VALUE
   FATAL_ERROR_PARAM("TEMP","parameter not set",LOCATION);
ENDIF_NO_VALUE

CAIRO_SET_PARAM("gmc_1","TEMP",temp) ;
CAIRO_SET_PARAM("gmc_2","TEMP",temp) ;
CAIRO_SET_PARAM("gmc_3","TEMP",temp) ;


//--------------------
// -------- vdd et vss
// ------------------

CAIRO_GET_GLOBAL_VARIABLE("VDD",vdd) ;
CAIRO_GET_GLOBAL_VARIABLE("VSS",vss) ;


// ---------------------------------------------------------------
// -------------------------- SIZE_FILTER_03 ----------------------
// ---------------------------------------------------------------

CAIRO_BEGIN_PROCEDURE("SIZE_FILTER_03")

  // ---------------------------------------------------------------
  // ----------------------------- Recuperation des parametres ------
  // ---------------------------------------------------------------

  // ------------------------------ Parametres communs

  //--------------------
  // -------- RESOLUTION
  // -------------------

  CAIRO_TRY_GET_VALUE("RESOLUTION",resolution)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("RESOLUTION : FILTER_03 specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //------------
  // -------- BW
  // -----------

  CAIRO_TRY_GET_VALUE("BW",bw)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("BW : FILTER_03 specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- OSR
  // ------------

  CAIRO_TRY_GET_VALUE("OSR",osr)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("OSR : FILTER_03 specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //-------------
  // -------- ASD
  // ------------

  CAIRO_TRY_GET_VALUE("ASD",asd)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("ASD : FILTER_03 specification is not set",LOCATION) ;
  ENDIF_NO_VALUE

  //--------------
  // -------- lmin
  // --------------

  CAIRO_TRY_GET_VALUE("LM1",lm1)
  IF_NO_VALUE
    FATAL_ERROR_PROCEDURE("LM1 : FILTER_03 specification is not set",LOCATION) ;
  ENDIF_NO_VALUE
```

```
CAIRO_TRY_GET_VALUE("LM3",lm3)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM3 : FILTER_O3 specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("LM5",lm5)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM5 : FILTER_O3 specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("LM7",lm7)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("LM7 : FILTER_O3 specification is not set",LOCATION) ;
ENDIF_NO_VALUE

//---------------------------------
// -------- effective gate voltages
// ---------------------------------

CAIRO_TRY_GET_VALUE("VEGM3",vegm3)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("VEGM3 : FILTER_O3 specification is not set",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEGM5",vegm5)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("VEGM5 : FILTER_O3 specification is not set",LOCATION) ;
ENDIF_NO_VALUE


// ----------------------------- GMC_1

//-----------------
// -------- RATIO_1
// ----------------

CAIRO_TRY_GET_VALUE("RATIO_1",ratio_1)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("RATIO_1: FILTER_O3 specification is not set for GMC_1",LOCATION) ;
ENDIF_NO_VALUE

//-----------------
// -------- AINT1_1
// ----------------

CAIRO_TRY_GET_VALUE("AINT1_1",aint1_1)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("AINT1_1 : FILTER_O3 specification is not set for GMC_1",LOCATION) ;
ENDIF_NO_VALUE

// ----------------------------- GMC_2

//-----------------
// -------- RATIO_2
// ----------------

CAIRO_TRY_GET_VALUE("RATIO_2",ratio_2)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("RATIO_2: FILTER_O3 specification is not set for GMC_2",LOCATION) ;
ENDIF_NO_VALUE

//-----------------
// -------- AINT1_2
// ----------------

CAIRO_TRY_GET_VALUE("AINT1_2",aint1_2)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("AINT1_2 : FILTER_O3 specification is not set for GMC_2",LOCATION) ;
ENDIF_NO_VALUE

// ----------------------------- GMC_3

//-----------------
// -------- RATIO_3
// ----------------

CAIRO_TRY_GET_VALUE("RATIO_3",ratio_3)
IF_NO_VALUE
  FATAL_ERROR_PROCEDURE("RATIO_3: FILTER_O3 specification is not set for GMC_3",LOCATION) ;
ENDIF_NO_VALUE

//-----------------
// -------- AINT1_3
// ----------------

CAIRO_TRY_GET_VALUE("AINT1_3",aint1_3)
IF_NO_VALUE
```

```
  FATAL_ERROR_PROCEDURE("AINT1_3 : FILTER_O3 specification is not set for GMC_3",LOCATION) ;
ENDIF_NO_VALUE


// ---------------------------------------------------------------
// ----------------------------- Contrainte facultative ----------
// ---------------------------------------------------------------

// Contrainte sur POWER_CONSUMPTION
EXAMINE_CONSTRAINT("POWER_CONSUMPTION")
  CASE_ABOVE              : GET_CONSTRAINT_ASSOCIATED_VALUES("POWER_CONSUMPTION",max_power) ;
                            power_constraint = true ;
                            break ;
  CASE_NONE               : break ;
  CASE_DEFAULT            : CAIRO_ERROR_MESSAGE("Unsupported constraint for POWER_CONSUMPTION in procedure "
                                                "SIZE_FILTER_O3 of %s\n",LOCATION.c_str()) ;
                            break ;
END_EXAMINE

// ---------------------------------------------------------------
// ----------------------------- Positionnements -----------------
// ---------------------------------------------------------------

// ----------------------------- GMC_1

CAIRO_SET_PARAM("gmc_1","RATIO",ratio_1) ;
CAIRO_SET_PARAM("gmc_1","RESOLUTION",resolution) ;
CAIRO_SET_PARAM("gmc_1","BW",bw) ;
CAIRO_SET_PARAM("gmc_1","AINT1",aint1_1) ;
CAIRO_SET_PARAM("gmc_1","OSR",osr) ;
CAIRO_SET_PARAM("gmc_1","ASD",asd) ;
CAIRO_SET_PARAM("gmc_1","VEGM3",vegm3) ;
CAIRO_SET_PARAM("gmc_1","VEGM5",vegm5) ;
CAIRO_SET_PARAM("gmc_1","LM1",lm1) ;
CAIRO_SET_PARAM("gmc_1","LM3",lm3) ;
CAIRO_SET_PARAM("gmc_1","LM5",lm5) ;
CAIRO_SET_PARAM("gmc_1","LM7",lm7) ;

// ----------------------------- GMC_2

CAIRO_SET_PARAM("gmc_2","RATIO",ratio_2) ;
CAIRO_SET_PARAM("gmc_2","RESOLUTION",resolution) ;
CAIRO_SET_PARAM("gmc_2","BW",bw) ;
CAIRO_SET_PARAM("gmc_2","AINT1",aint1_2) ;
CAIRO_SET_PARAM("gmc_2","OSR",osr) ;
CAIRO_SET_PARAM("gmc_2","ASD",asd) ;
CAIRO_SET_PARAM("gmc_2","VEGM3",vegm3) ;
CAIRO_SET_PARAM("gmc_2","VEGM5",vegm5) ;
CAIRO_SET_PARAM("gmc_2","LM1",lm1) ;
CAIRO_SET_PARAM("gmc_2","LM3",lm3) ;
CAIRO_SET_PARAM("gmc_2","LM5",lm5) ;
CAIRO_SET_PARAM("gmc_2","LM7",lm7) ;

// ----------------------------- GMC_3

CAIRO_SET_PARAM("gmc_3","RATIO",ratio_3) ;
CAIRO_SET_PARAM("gmc_3","RESOLUTION",resolution) ;
CAIRO_SET_PARAM("gmc_3","BW",bw) ;
CAIRO_SET_PARAM("gmc_3","AINT1",aint1_3) ;
CAIRO_SET_PARAM("gmc_3","OSR",osr) ;
CAIRO_SET_PARAM("gmc_3","ASD",asd) ;
CAIRO_SET_PARAM("gmc_3","VEGM3",vegm3) ;
CAIRO_SET_PARAM("gmc_3","VEGM5",vegm5) ;
CAIRO_SET_PARAM("gmc_3","LM1",lm1) ;
CAIRO_SET_PARAM("gmc_3","LM3",lm3) ;
CAIRO_SET_PARAM("gmc_3","LM5",lm5) ;
CAIRO_SET_PARAM("gmc_3","LM7",lm7) ;


// ---------------------------------------------------------------
// ----------------------------- Dimensionnements ----------------
// ---------------------------------------------------------------

loop_size_filter_o3:

// ----------------------------- GMC_1

RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    CAIRO_COMPUTE("gmc_1","SIZE_GMC") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 1  :
```

```
          case 2  :
          case 3  :
          case 4  :
          case 5  :
          case 6  :
          case 7  :
          case 8  : CAIRO_SET_VALUE("CAPINT_1",capint_1,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_1",ibias_1,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_1,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_1,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_1,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_1",modulation_coeff_1,CP_NAV) ;

                    CAIRO_SET_VALUE("CAPINT_2",capint_2,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_2",ibias_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_2,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_2",modulation_coeff_2,CP_NAV) ;

                    CAIRO_SET_VALUE("CAPINT_3",capint_3,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_3",ibias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_3",modulation_coeff_3,CP_NAV) ;

                    CAIRO_SET_VALUE("POWER_CONSUMPTION",power_consumption,CP_NAV) ;

                    CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;

                    goto bad_return_size_filter_o3 ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for SIZE_GMC in GMC_1\n",PROCEDURE_STATUS) ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("gmc_1","CAPINT",capint_1) ;
CAIRO_GET_PARAM("gmc_1","IBIAS",ibias_1) ;
CAIRO_GET_PARAM("gmc_1","VBC",vbc_1) ;
CAIRO_GET_PARAM("gmc_1","VCP",vcp_1) ;
CAIRO_GET_PARAM("gmc_1","VBIAS",vbias_1) ;
CAIRO_GET_PARAM("gmc_1","MODULATION_COEFF",modulation_coeff_1) ;

// ---------------------------- GMC_2

RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    CAIRO_COMPUTE("gmc_2","SIZE_GMC") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 1  :
          case 2  :
          case 3  :
          case 4  :
          case 5  :
          case 6  :
          case 7  :
          case 8  : CAIRO_SET_VALUE("CAPINT_2",capint_2,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_2",ibias_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_2,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_2,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_2",modulation_coeff_2,CP_NAV) ;

                    CAIRO_SET_VALUE("CAPINT_3",capint_3,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_3",ibias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_3",modulation_coeff_3,CP_NAV) ;

                    CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;

                    CAIRO_SET_VALUE("POWER_CONSUMPTION",power_consumption,CP_NAV) ;

                    goto bad_return_size_filter_o3 ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for SIZE_GMC in GMC_2\n",PROCEDURE_STATUS) ;
```

```
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("gmc_2","CAPINT",capint_2) ;
CAIRO_GET_PARAM("gmc_2","IBIAS",ibias_2) ;
CAIRO_GET_PARAM("gmc_2","VBC",vbc_2) ;
CAIRO_GET_PARAM("gmc_2","VCP",vcp_2) ;
CAIRO_GET_PARAM("gmc_2","VBIAS",vbias_2) ;
CAIRO_GET_PARAM("gmc_2","MODULATION_COEFF",modulation_coeff_2) ;

// ---------------------------- GMC_3

RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    CAIRO_COMPUTE("gmc_3","SIZE_GMC") ;

    ON_PROCEDURE_STATUS

      switch(PROCEDURE_STATUS)
        {
          case 1  :
          case 2  :
          case 3  :
          case 4  :
          case 5  :
          case 6  :
          case 7  :
          case 8  : CAIRO_SET_VALUE("CAPINT_3",capint_3,CP_NAV) ;
                    CAIRO_SET_VALUE("IBIAS_3",ibias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBC",vbc_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VCP",vcp_3,CP_NAV) ;
                    CAIRO_SET_VALUE("VBIAS",vbias_3,CP_NAV) ;
                    CAIRO_SET_VALUE("MODULATION_COEFF_3",modulation_coeff_3,CP_NAV) ;

                    CAIRO_SET_RETURN_PROCEDURE(PROCEDURE_STATUS) ;

                    CAIRO_SET_VALUE("POWER_CONSUMPTION",power_consumption,CP_NAV) ;

                    goto bad_return_size_filter_o3 ;
                    break ;

          default : CAIRO_ERROR_MESSAGE("Unknown status %li for SIZE_GMC in GMC_3\n",PROCEDURE_STATUS) ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_GET_PARAM("gmc_3","CAPINT",capint_3) ;
CAIRO_GET_PARAM("gmc_3","IBIAS",ibias_3) ;
CAIRO_GET_PARAM("gmc_3","VBC",vbc_3) ;
CAIRO_GET_PARAM("gmc_3","VCP",vcp_3) ;
CAIRO_GET_PARAM("gmc_3","VBIAS",vbias_3) ;
CAIRO_GET_PARAM("gmc_3","MODULATION_COEFF",modulation_coeff_3) ;


// Calcul de la consommation
power_consumption = 6.0 * (ibias_1 * ibias_2 * ibias_3) * vdd ;


// Boucle facultative sur la consommation

if(power_constraint == true)
  {
    if(power_consumption > max_power)
      {
        if(current_ratio2 > min_ratio)
          {
            if(current_ratio3 <= min_ratio)
              {
                current_ratio2 = current_ratio2 - ratio_step ;
                current_ratio3 = current_ratio2 ;

                CAIRO_SET_PARAM("gmc_1","RATIO",current_ratio1) ;
                CAIRO_SET_PARAM("gmc_2","RATIO",current_ratio2) ;
                CAIRO_SET_PARAM("gmc_3","RATIO",current_ratio3) ;

                goto loop_size_filter_o3 ;
              }
            else
              {
                current_ratio3 = current_ratio3 - ratio_step ;

                CAIRO_SET_PARAM("gmc_1","RATIO",current_ratio1) ;
                CAIRO_SET_PARAM("gmc_3","RATIO",current_ratio3) ;
```

```
                goto loop_size_filter_o3 ;
              }
          }
        else
          {
            CAIRO_SET_VALUE("CAPINT_1",capint_1,CP_VALID) ;
            CAIRO_SET_VALUE("IBIAS_1",ibias_1,CP_VALID) ;
            CAIRO_SET_VALUE("VBC",vbc_1,CP_VALID) ;
            CAIRO_SET_VALUE("VCP",vcp_1,CP_VALID) ;
            CAIRO_SET_VALUE("VBIAS",vbias_1,CP_VALID) ;
            CAIRO_SET_VALUE("MODULATION_COEFF_1",modulation_coeff_1,CP_VALID) ;

            CAIRO_SET_VALUE("CAPINT_2",capint_2,CP_VALID) ;
            CAIRO_SET_VALUE("IBIAS_2",ibias_2,CP_VALID) ;
            CAIRO_SET_VALUE("VBC",vbc_2,CP_VALID) ;
            CAIRO_SET_VALUE("VCP",vcp_2,CP_VALID) ;
            CAIRO_SET_VALUE("VBIAS",vbias_2,CP_VALID) ;
            CAIRO_SET_VALUE("MODULATION_COEFF_2",modulation_coeff_2,CP_VALID) ;

            CAIRO_SET_VALUE("CAPINT_3",capint_3,CP_VALID) ;
            CAIRO_SET_VALUE("IBIAS_3",ibias_3,CP_VALID) ;
            CAIRO_SET_VALUE("VBC",vbc_3,CP_VALID) ;
            CAIRO_SET_VALUE("VCP",vcp_3,CP_VALID) ;
            CAIRO_SET_VALUE("VBIAS",vbias_3,CP_VALID) ;
            CAIRO_SET_VALUE("MODULATION_COEFF_3",modulation_coeff_3,CP_VALID) ;

            CAIRO_SET_VALUE("POWER_CONSUMPTION",power_consumption,CP_TOO_LARGE) ;

            CAIRO_SET_RETURN_PROCEDURE(25) ;

            goto bad_return_size_filter_o3 ;
          }
      }
    else
      {
        goto good_return_size_filter_o3 ;
      }
  }

  // ---------------------------------------------------------------
  // ---------------------------- Retour des valeurs ---------------
  // ---------------------------------------------------------------

  good_return_size_filter_o3:

  CAIRO_SET_VALUE("POWER_CONSUMPTION",power_consumption,CP_VALID) ;

  CAIRO_SET_VALUE("VBC",vbc_1,CP_VALID) ;
  CAIRO_SET_VALUE("VCP",vcp_1,CP_VALID) ;
  CAIRO_SET_VALUE("VBIAS",vbias_1,CP_VALID) ;

  CAIRO_SET_VALUE("CAPINT_1",capint_1,CP_VALID) ;
  CAIRO_SET_VALUE("IBIAS_1",ibias_1,CP_VALID) ;
  CAIRO_SET_VALUE("MODULATION_COEFF_1",modulation_coeff_1,CP_VALID) ;

  CAIRO_SET_VALUE("CAPINT_2",capint_2,CP_VALID) ;
  CAIRO_SET_VALUE("IBIAS_2",ibias_2,CP_VALID) ;
  CAIRO_SET_VALUE("MODULATION_COEFF_2",modulation_coeff_2,CP_VALID) ;

  CAIRO_SET_VALUE("CAPINT_3",capint_3,CP_VALID) ;
  CAIRO_SET_VALUE("IBIAS_3",ibias_3,CP_VALID) ;
  CAIRO_SET_VALUE("MODULATION_COEFF_3",modulation_coeff_3,CP_VALID) ;

  CAIRO_SET_RETURN_PROCEDURE(CP_OK) ;

  return ;


  bad_return_size_filter_o3:
  return ;

END_PROCEDURE


CAIRO_DEFAULT_PROCEDURE
  FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"Unknown procedure",LOCATION) ;
END_DEFAULT_PROCEDURE

CAIRO_END_DSES(filter_o3)


// ------------------------------------------------------------------------------------------------
// ------------------------------------------- LAYOUT -------------------------------------------
// ------------------------------------------------------------------------------------------------

CAIRO_BEGIN_LAYOUT(filter_o3,char *name)
```

```
  double height = 0.0 ;
  double width  = 0.0 ;

  height = CAIRO_HEIGHT(name) ;
  width  = CAIRO_WIDTH(name)  ;
CAIRO_END_LAYOUT(filter_o3)
```

Annexe I

Code source du fichier de tests du filtre en $0,13\ \mu m$

```
/*****************************************************************************/
/*                        Fichier : testFILTER_03.cpp                        */
/*                                                                           */
/* Description : fichier de tests du filtre                                  */
/*                                                                           */
/* Langage : C++                Version : 1.0              Date : 08/11/2005 */
/* Auteur  : The Masked Cucumber                Qualif: The Masked Cucumber  */
/*                                                                           */
/* License : QPL                                                             */
/*                                                                           */
/* Historique                                                                */
/* 1.0   - 08/11/2005 : version initiale                                     */
/*                                                                           */
/* Fonctions :                                                               */
/*                                                                           */
/*****************************************************************************/

#include <iostream>
#include <fstream>

using namespace std ;

#include "cairoplus.h"


int main(int argc, char *argv[])
{
  double resolution         = 8.0      ;
  double bw                 = 50.0e03  ;
  double ratio_1            = 1.0      ;
  double ratio_2            = 1.0      ;
  double ratio_3            = 1.0      ;
  double aint1_1            = 0.0333   ;
  double aint1_2            = 0.5      ;
  double aint1_3            = 0.5      ;
  double osr                = 64.0     ;
  double asd                = 0.5      ;
  double vdd                = 1.0      ;
  double vss                = 0.0      ;
  double vegm3              = 0.106    ;
  double vegm5              = -0.1     ;
  double lm1                = 3.5e-06  ;
  double lm3                = 2.8e-06  ;
  double lm5                = 2.8e-06  ;
  double lm7                = 3.5e-06  ;

  double power_consumption  = 0.0         ;
  //double max_power          = 0.52484e-16 ;

  double vbc                = 0.0 ;
  double vcp                = 0.0 ;
  double vbias              = 0.0 ;

  double capint_1           = 0.0 ;
  double ibias_1            = 0.0 ;
  double modulation_coeff_1 = 0.0 ;
```

```
double capint_2           = 0.0 ;
double ibias_2            = 0.0 ;
double modulation_coeff_2 = 0.0 ;

double capint_3           = 0.0 ;
double ibias_3            = 0.0 ;
double modulation_coeff_3 = 0.0 ;

// Feedback DAC RZ Characteristics
double tau                = 0.75      ;

// Differential feedback coefficients
double coeff1             = 0.5  /tau ;
double coeff2             = 0.1  /tau ;
double coeff3             = 0.11 /tau ;


cout << "--- Parametres ------------" << endl ;
cout << "RATIO_1     = " << ratio_1     << endl ;
cout << "RATIO_2     = " << ratio_2     << endl ;
cout << "RATIO_3     = " << ratio_3     << endl ;
cout << "RESOLUTION  = " << resolution  << endl ;
cout << "BW          = " << bw          << endl ;
cout << "Aint1_1     = " << aint1_1     << endl ;
cout << "Aint1_2     = " << aint1_2     << endl ;
cout << "Aint1_3     = " << aint1_3     << endl ;
cout << "OSR         = " << osr         << endl ;
cout << "VDD         = " << vdd         << endl ;
cout << "VSS         = " << vss         << endl ;
cout << "VEGM3       = " << vegm3       << endl ;
cout << "VEGM5       = " << vegm5       << endl ;
cout << "Lm1         = " << lm1         << endl ;
cout << "Lm3         = " << lm3         << endl ;
cout << "Lm5         = " << lm5         << endl ;
cout << "Lm7         = " << lm7         << endl ;
cout << "---------------------------" << endl ;


CAIRO_NEW_CHIP("test_filter_o3") ;

// --------
// Creation
// --------

CAIRO_CREATE("libFILTER_O3","filter_o3","filter_o3") ;

CAIRO_SET_GLOBAL_VARIABLE("VDD",vdd) ;
CAIRO_SET_GLOBAL_VARIABLE("VSS",vss) ;

// ----
// DSES
// ----

CAIRO_SET_PARAM("filter_o3","TEMP"      ,300.15    ) ;
CAIRO_SET_PARAM("filter_o3","RESOLUTION",resolution) ;
CAIRO_SET_PARAM("filter_o3","BW"        ,bw        ) ;
CAIRO_SET_PARAM("filter_o3","RATIO_1"   ,ratio_1   ) ;
CAIRO_SET_PARAM("filter_o3","RATIO_2"   ,ratio_2   ) ;
CAIRO_SET_PARAM("filter_o3","RATIO_3"   ,ratio_3   ) ;
CAIRO_SET_PARAM("filter_o3","AINT1_1"   ,aint1_1   ) ;
CAIRO_SET_PARAM("filter_o3","AINT1_2"   ,aint1_2   ) ;
CAIRO_SET_PARAM("filter_o3","AINT1_3"   ,aint1_3   ) ;
CAIRO_SET_PARAM("filter_o3","OSR"       ,osr       ) ;
CAIRO_SET_PARAM("filter_o3","ASD"       ,asd       ) ;
CAIRO_SET_PARAM("filter_o3","VEGM3"     ,vegm3     ) ;
CAIRO_SET_PARAM("filter_o3","VEGM5"     ,vegm5     ) ;
CAIRO_SET_PARAM("filter_o3","LM1"       ,lm1       ) ;
CAIRO_SET_PARAM("filter_o3","LM3"       ,lm3       ) ;
CAIRO_SET_PARAM("filter_o3","LM5"       ,lm5       ) ;
CAIRO_SET_PARAM("filter_o3","LM7"       ,lm7       ) ;

// Contrainte pour POWER_CONSUMPTION
//CAIRO_SET_PARAM_CONSTRAINT("filter_o3","POWER_CONSUMPTION",CP_ABOVE,max_power) ;

RESET_PROCEDURE_STATUS ;

while(PROCEDURE_STATUS != CP_OK)
  {
    CAIRO_COMPUTE("filter_o3","SIZE_FILTER_O3") ;

    ON_PROCEDURE_STATUS
      switch(PROCEDURE_STATUS)
        {
          case 1  :
          case 2  :
          case 3  :
          case 4  :
          case 5  :
```

```
          case 6  :
          case 8  :
          case 9  :
          case 10 :
          case 11 :
          case 12 :
          case 13 :
          case 14 :
          case 15 :
          case 16 :
          case 17 :
          case 18 :
          case 19 :
          case 20 :
          case 21 :
          case 22 :
          case 23 :
          case 24 : CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("filter_o3",PROCEDURE_STATUS).c_str()) ;
                    CAIRO_ERROR_MESSAGE("No loop available\n") ;
                    break ;
          default : CAIRO_MESSAGE(CAIRO_TRANSLATE_MESSAGE_PROCEDURE("filter_o3",PROCEDURE_STATUS).c_str()) ;
                    CAIRO_ERROR_MESSAGE("No loop available\n") ;
                    break ;
        }

    END_ON_PROCEDURE_STATUS
  }

CAIRO_RESHAPE() ;

// CAIRO_STABILITY();

CAIRO_GET_PARAM("filter_o3","POWER_CONSUMPTION" ,power_consumption ) ;

CAIRO_GET_PARAM("filter_o3","VBC"               ,vbc               ) ;
CAIRO_GET_PARAM("filter_o3","VCP"               ,vcp               ) ;
CAIRO_GET_PARAM("filter_o3","VBIAS"             ,vbias             ) ;

CAIRO_GET_PARAM("filter_o3","CAPINT_1"          ,capint_1          ) ;
CAIRO_GET_PARAM("filter_o3","IBIAS_1"           ,ibias_1           ) ;
CAIRO_GET_PARAM("filter_o3","MODULATION_COEFF_1",modulation_coeff_1) ;

CAIRO_GET_PARAM("filter_o3","CAPINT_2"          ,capint_2          ) ;
CAIRO_GET_PARAM("filter_o3","IBIAS_2"           ,ibias_2           ) ;
CAIRO_GET_PARAM("filter_o3","MODULATION_COEFF_2",modulation_coeff_2) ;

CAIRO_GET_PARAM("filter_o3","CAPINT_3"          ,capint_3          ) ;
CAIRO_GET_PARAM("filter_o3","IBIAS_3"           ,ibias_3           ) ;
CAIRO_GET_PARAM("filter_o3","MODULATION_COEFF_3",modulation_coeff_3) ;

{
  // Generation automatique du fichier de simulation

  ofstream cirFile ;

  double input_amp       = (1.0 / 2.0) * modulation_coeff_1 * ibias_1 * asd ;
  double feedback_coeff1 = modulation_coeff_1 * ibias_1 * coeff1 ;
  double feedback_coeff2 = modulation_coeff_2 * ibias_2 * coeff2 ;
  double feedback_coeff3 = modulation_coeff_3 * ibias_3 * coeff3 ;
  double fs              = 2.0 * bw * osr  ;
  double Ts              = 1.0 / fs        ;
  double N               = 16384.0         ;
  double L               = 1.2 * N         ;
  double fin             = (13.0 / N) * fs ;
  double real_tau        = tau * Ts        ;
  double real_td         = Ts - real_tau   ;

  cirFile.open("3rd_sd_tran_analysis.cir",ios::out) ;

  if(cirFile == NULL)
    {
      cerr << "Problem when opening data file 3rd_sd_tran_analysis.cir" << endl ;
      exit(1) ;
    }

  cirFile << "* 3rd Order Sigma-Delta Simulation *\n" ;
  cirFile << "\n" ;
  cirFile << "*-----------------------------------*\n" ;
  cirFile << "\n" ;

  cirFile << "*-----------------------------------*\n" ;
  cirFile << "*    SUPPLY AND BIASING           *\n" ;
  cirFile << "*-----------------------------------*\n" ;
  cirFile << "vss ss 0 0.0 \n" ;
  cirFile << "vdd dd ss " << vdd  << endl ;
  cirFile << "vbc bc ss " << vbc << endl ;
  cirFile << "vcp cp ss " << vcp << endl ;
  cirFile << "ibias bias  ss " << ibias_1 << endl ;
```

```
cirFile << "voutp outpp ss " << vdd / 2.0 << endl ;
cirFile << "voutn outnn ss " << vdd / 2.0 << endl ;
cirFile << "cn1 ind1 0 " << capint_1 << endl ;
cirFile << "cp1 ipd1 0 " << capint_1 << endl ;
cirFile << "cn2 ind2 0 " << capint_2 << endl ;
cirFile << "cp2 ipd2 0 " << capint_2 << endl ;
cirFile << "cn3 ind3 0 " << capint_3 << endl ;
cirFile << "cp3 ipd3 0 " << capint_3 << endl ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  Input Signal                    *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "iina ipd1 0 sin(0 " << input_amp  << " " << fin << " 0 0 )\n"  ;
cirFile << "iinb ind1 0 sin(0 " << -input_amp << " " << fin << " 0 0 )\n"  ;

cirFile << "vinput input 0 sin(0 -1.0 " << fin << " 0 0 )\n" ;

cirFile << "*------------------------------*\n" ;
cirFile << "*  The first integrator        *\n" ;
cirFile << "*------------------------------*\n" ;
cirFile << ".include filter_o3_gmc_1.spi\n" ;
cirFile << "Xint1  ss dd bias cp bc outp1 outn1 ipd1 ind1 filter_o3_gmc_1\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  The Second Integrator           *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << ".include filter_o3_gmc_2.spi\n" ;
cirFile << "Xint2  ss dd bias cp bc outp2 outn2 ipd2 ind2 filter_o3_gmc_2\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  The Third Integrator            *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << ".include filter_o3_gmc_3.spi\n" ;
cirFile << "Xint3  ss dd bias cp bc outp3 outn3 ipd3 ind3 filter_o3_gmc_3\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  Connections                     *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "v1 outp1 ipd2 0\n" ;
cirFile << "v2 outn1 ind2 0\n" ;
cirFile << "v3 outp2 ipd3 0\n" ;
cirFile << "v4 outn2 ind3 0\n" ;
cirFile << "v5 outp3 outpp 0\n" ;
cirFile << "v6 outn3 outnn 0\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  Sample and Hold                 *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "ha nout 0 voutp 1\n" ;
cirFile << "hb pout 0 voutn 1\n" ;
cirFile << ".model sto modfas fs=" << fs << " tacq=10p dv=0\n" ;
cirFile << "ysamn SA_HO pin: nout  0 no_ut 0 model: sto\n" ;
cirFile << "ysamp SA_HO pin: pout  0 po_ut 0 model: sto\n" ;
cirFile << "ysami SA_HO pin: input 0 in_ut 0 model: sto\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  COMPARATOR                      *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "compd1 po_ut no_ut digoutp digoutn vhi=1 vlo=-1 tcom=1p\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  P Feedback DAC model            *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "gina1 0 to_ina1 fbn 0 " << feedback_coeff1 << endl ;
cirFile << "vgina1 to_ina1 ind1 0\n" ;
cirFile << "gina2 0 to_ina2 fbn 0 " << feedback_coeff2 << endl ;
cirFile << "vgina2 to_ina2 ind2 0\n" ;
cirFile << "gina3 0 to_ina3 fbn 0 " << feedback_coeff3 << endl ;
cirFile << "vgina3 to_ina3 ind3 0\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  N Feedback DAC model            *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << "ginb1 0 to_inb1 fbp 0 " << feedback_coeff1 << endl ;
cirFile << "vginb1 to_inb1 ipd1 0\n" ;
cirFile << "ginb2 0 to_inb2 fbp 0 " << feedback_coeff2 << endl ;
cirFile << "vginb2 to_inb2 ipd2 0\n" ;
cirFile << "ginb3 0 to_inb3 fbp 0 " << feedback_coeff3 << endl ;
cirFile << "vginb3 to_inb3 ipd3 0\n" ;

cirFile << "*----------------------------------*\n" ;
cirFile << "*  INPUT PARAMETES                 *\n" ;
cirFile << "*----------------------------------*\n" ;
cirFile << ".param fs  = " <<  fs       << endl ;
cirFile << ".param Ts  = " <<  Ts       << endl ;
cirFile << ".param N   = " <<  N        << endl ;
cirFile << ".param L   = " <<  L        << endl ;
cirFile << ".param fin = " <<  fin      << endl ;
```

```
    cirFile << ".param tau = " <<  real_tau << endl ;
    cirFile << ".param td  = " <<  real_td  << endl ;
    cirFile << ".param tr  = 5e-10\n" ;
    cirFile << ".param tf  = 5e-10\n" ;

    cirFile << "*-------------------------------------\n" ;
    cirFile << "*             PULSE                   \n" ;
    cirFile << "*-------------------------------------\n" ;
    cirFile << "vp pul 0 pulse (0 1 td tr tf tau Ts)\n" ;

    cirFile << "*-------------------------------------\n" ;
    cirFile << "*             MULTIPLIER              \n" ;
    cirFile << "*-------------------------------------\n" ;
    cirFile << "ymulp mult digoutp pul fbp\n" ;
    cirFile << "ymuln mult digoutn pul fbn\n" ;

    cirFile << "*------------------------------\n" ;
    cirFile << "*  ANALYSIS                    \n" ;
    cirFile << "*------------------------------\n" ;
    cirFile << ".op\n" ;
    cirFile << ".plot tran v(digoutp)\n" ;
    cirFile << ".print tran v(digoutp)\n" ;
    cirFile << "*.plot tran i(v1) i(v2)\n" ;
    cirFile << "*.plot tran i(v3) i(v4)\n" ;
    cirFile << "*.plot tran i(v5) i(v6)\n" ;
    cirFile << "*.plot tran i(vdacp1) i(vdacn1)\n" ;
    cirFile << "*.plot tran i(vdacp2) i(vdacn2)\n" ;
    cirFile << "*.plot tran i(vdacp3) i(vdacn3)\n" ;
    cirFile << "*.plot tran v(RZP) v(PA) v(PB)\n" ;
    cirFile << "*.plot tran v(RZN) v(NA) v(NB)\n" ;

    cirFile << ".TRAN " << Ts << " " << L * Ts << " " << Ts << endl ;
    cirFile << ".four label=o1 v(digoutp)\n" ;
    cirFile << ".four label=i1 v(in_ut) \n" ;
    cirFile << ".optfour \n" ;
    cirFile << "+ nbpt = " << N << endl ;
    cirFile << "+ interpolate = 0\n" ;
    cirFile << "+ window = blackman\n" ;
    cirFile << "+ display_input = 0 \n" ;
    cirFile << "+ normalized = 1\n" ;
    cirFile << "+ fnormal = " << fin << endl ;
    cirFile << "+ tstop = " <<  L * Ts << endl ;
    cirFile << "+ tstart = " << (L * Ts) - (N * Ts) << endl ;
    cirFile << "+ fs = " << fs << endl ;
    cirFile << ".plot four fourdb(o1) fourdb(i1)\n" ;
    cirFile << ".print four fourdb(o1) fourdb(i1)\n" ;
    cirFile << ".option eps=1e-6 be hmax=0.01u nowavecomplex\n" ;

    cirFile << "* --------------------------------*\n" ;
    cirFile << "* Technologie                     *\n" ;
    cirFile << "* --------------------------------*\n" ;
    cirFile << "\n" ;
    cirFile << "\n" ;
    cirFile << ".PARAM MOSLL_DEV=0\n" ;
    cirFile << "\n" ;
    cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_poly.lib PRO_stat\n" ;
    cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_active.lib PRO_stat\n" ;
    cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/common_go1.lib PRO_stat\n" ;
    cirFile << ".LIB /users/soft/techno/dev/grenoble/hcmos9/modeles_mars03/mos_bsim3_LL.lib mosLL_stat\n" ;
    cirFile << "\n" ;
    cirFile << "\n" ;
    cirFile << ".end\n" ;
  }

  cout << "--- Final ----------------------------------------" << endl ;
  cout << "main : POWER_CONSUMPTION  = " << power_consumption  << endl ;

  cout << "main : VBC                = " << vbc                << endl ;
  cout << "main : VCP                = " << vcp                << endl ;
  cout << "main : VBIAS              = " << vbias              << endl ;

  cout << "main : CAPINT_1           = " << capint_1           << endl ;
  cout << "main : IBIAS_1            = " << ibias_1            << endl ;
  cout << "main : MODULATION_COEFF_1 = " << modulation_coeff_1 << endl ;

  cout << "main : CAPINT_2           = " << capint_2           << endl ;
  cout << "main : IBIAS_2            = " << ibias_2            << endl ;
  cout << "main : MODULATION_COEFF_2 = " << modulation_coeff_2 << endl ;

  cout << "main : CAPINT_3           = " << capint_3           << endl ;
  cout << "main : IBIAS_3            = " << ibias_3            << endl ;
  cout << "main : MODULATION_COEFF_3 = " << modulation_coeff_3 << endl ;
  cout << "-------------------------------------------------" << endl ;

  CAIRO_SAVE_CHIP("test_filter_o3") ;

  return 0 ;
}
```

Annexe J

Publications

Placement Optimal d'Objets Déformables dans l'Environnement de Conception Analogique CAIRO+
Pierre Nguyen Tuong, Marie-Minerve Louërat, Alain Greiner
Troisième Colloque du GDR CAO de circuits et systèmes intégrés, Paris, France, Mai 2002, pages 29-32

Analog IP Cores Design with CAIRO+
Marie-Minerve Louërat and Pierre Nguyen Tuong and Alain Greiner
and Laurent de Lamarre and Vincent Bourguet
University Booth, Design Automation and Test in Europe, DATE 2003, Paris, France

Managing the Shape Function of Analog Devices in a Slicing Tree Floorplan
Pierre Nguyen Tuong and Marie-Minerve Louërat and Alain Greiner
Proceedings of the 11th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES 2004)
Szczecin, Poland, June 2004, pages 226-229

Guidelines for Designing Smart and Reusable Analog IP Cores
Pierre Nguyen Tuong and Marie-Minerve Louërat and Alain Greiner
Sophia Antipolis MicroElectronics Forum (SAME 2004)

A Language to Design generators of Analog Functions
Pierre Nguyen Tuong and Vincent Bourguet and Laurent de Lamarre and Marie-Minerve Louërat and Alain Greiner
Forum on Specification & Design Languages (FDL'04), Lille, France, September 2004, pages 30–31

Synthèse d'un IP amplificateur analogique CMOS avec CAIRO+
Ramy Iskander, Laurent de Lamarre, Pierre Nguyen Tuong,
Marie-Minerve Louërat, Andreas Kaiser
6ème Colloque sur le Traitement Analogique de l'Information
du Signal et ses Applications (TAISA'2005)
Marseille, France, octobre 2005, pages 69–72

Bibliographie

[Abo02] Hassan Aboushady. *Conception en Vue de la Réutilisation de Convertisseurs Analogique-Numérique ΣΔ Temps-Continu Mode-Courant.* PhD thesis, Université Pierre et Marie Curie, Paris VI, January 2002.

[ANA92] ANACAD-EES. *ELDO-FAS Dynamical System Modeling user's manual, version 4.1.x.* 1992.

[ANA94] ANACAD-EES. *HDL-A language reference manual.* 1994.

[AVL62] G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. In *Soviet Mathematics*, volume 3, pages 1259–1263. Dokladi Akademia Nauk SSSR, 1962.

[BV04] F. De Bernardinis and A. Sangiovanni Vincentelli. A Methodology for System Level Analog Design Space Exploration. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, volume 1, pages 200–201. DATE-04, 2004.

[Cad] Cadence. Virtuoso Neocell/Virtuoso Neocircuit. http ://www.cadence.com.

[CCH+95] Yuhua Cheng, Mansun Chan, Kelvin Hui, Min chie Jeng, Zhihong Liu, Jianhui Huang, Kai Chen, James Chen, Robert Tu, Ping K. Ko, and Chenming Hu. *BSIM3v3 Manual.* University of California, Berkeley, 1995.

[CGRC91] J.M. Cohn, D.J. Garrod, R.A. Rutenbar, and L.R. Carley. KOAN/ANAGRAM II : New Tools for Device-Level Analog Placement and Routing. *IEEE Journal of Solid State Circuits*, 26 :330–342, march 1991.

[CH99] Yuha CHENG and Chenming Hu. *MOSFET Modeling and BSIM3 User's Guide.* Kluwer Academic Publishers, 1999.

[Des] Barcelona Design. Barcelona design. http ://www.barcelonadesign.com.

[Des01] Mohamed Dessouky. *Design for Reuse of Analog Circuits. Case Study :Very Low-Voltage Delta-Sigma Modulator.* PhD thesis, Université Pierre et Marie Curie, Paris VI, January 2001.

[DL00] Mohamed Dessouky and Marie-Minerve Louërat. A Layout Approach for Electrical and Physical Design Integration of High-Performance Analog Circuits. pages 291–298, San Jose, USA, 2000. ISQED'00.

[dPDL+01] Geert Van der Plas, Geert Debyser, Francky Leyn, Koen Lampaert, Jan Vandenbussche, George G.E. Gielen, Willy Sansen, Petar Veselinovic, and D. Leenaerts. AMGIE-A Synthesis Environment for CMOS Analog Integrated Circuits. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 20(9) :1037–1057, september 2001.

[DW86] C. Liu D. Wong. A new algorithm for floorplan design. In *Proceedings of the IEEE/ACM Design Automation conference*. IEEE/ACM, IEE, 1986.

[EDA01] EDA. Verilog-AMS Sample Library. http ://www.eda.org/verilog-ams/models/opamp.va, 2001.

[Fa94] E. Felt and al. Measurement and Modeling of MOS Transistor Current Mismatch in Analog IC's. In *Proceedings of the IEEE/ACM Design Automation conference*, pages 272–277. ACM, 1994.

[Ga92] A. Greiner and al. Alliance : A complete set of CAD tools for teaching VLSI Design. In *EuroChip Workshop on VLSI Design Training*, pages 230–237, 1992.

[Gog99] Eric Goguey. Dico.fr. http ://www.dicofr.com, 1999.

[GR00] G. Gielen and K. Rutenbar. Computer aided design of analog and mixed-signal integrated circuits. *Proc IEEE*, 88 :1825–1852, december 2000.

[Gra03] Mentor Graphics. *Eldo User's Manual, Software Version 6.0_1, Release 2003.1*. 2003.

[Has01] Hastings. *The Art of Analog Layout*. Prentice Hall, 2001.

[Her] Yannick Herve. Cours d'electronique numérique et de méthodologie de CAO Electronique VHDL-AMS et conception système. http ://www-ensps.u-strasbg.fr/CoursEN/option3A/vhdlamsb.htm.

[HHH+97] J. Hanna, R. Hillman, H. Hirsch, T. Noh, and R. Vemuri. *Using WAVES and* VHDL *for Effective Design and Testing*. Kluwer Academic Publishers, 1997.

[HRC89] R. Harjani, Rutenbar R, and L.R. Carley. OASYS : A framework for analog circuit synthesis. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systsems*, 8 :1247–1266, december 1989.

[IF90] M. Ismail and J. Franca. *Introduction to Analog* VLSI *Design Automation*. Kluwer Academic Publishers, The Netherlands, 1990.

[Inc01] Cadence Design Systems Inc. *SKILL language reference. Product version06.00*. 2001.

[JGSV90] Gani Jusuf, Paul R. Gray, and Alberto Sangiovanni-Vincentelli. CADICS - Cyclic Analog-To-Digital Converter Synthesis. In *Proc. IEEE Int. on Computer-Aided-Design*, pages 286–289. CAD, IEEE, november 1990.

[JT92] C. Candy James and Gabor C. Temes. *Oversampling Delta-Sigma Data Convertors*. IEEE press, 1992.

[Kay03] M. Kayal. Procedural Analog Design (PAD). http ://legwww.epfl.ch/CSL/PAD/PAD_pages/DesignToolHome.html, 2003.

[KSG90] H. Koh, C. Séquin, and P. Gray. OPASYN : A compiler for CMOS operationnal amplifiers. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systsems*, 9 :113–125, february 1990.

[LGS99] K. Lampaert, G. Gielen, and W.M. Sansen. *Analog Layout Generation for Performance and Manufacturability*. Kluwer Academic Publishers, The Netherlands, 1999.

[Lém95] François Lémery. *Modélisation comportementale des circuits analogiques et mixtes*. PhD thesis, Institut National Polytechnique de Grenoble, Decembre 1995.

[Mar] Kenneth Martin. Ballistic. http ://www.eecg.toronto.edu/ gdt/HTML /Main.html.

[Mik81] H. Mikoshiba. Stress Properties of Silicon-Gate MOS Devices. In *Solid-State Electronics*, volume 24, pages 221–232. IEEE, 1981.

[MPVH95] F. Medeiro, Belen Pérez-Verdú, and J.L. Huertas. A Vertically Integrated Tool for Automated Design of $\Sigma\Delta$ Modulators. *Journal of Solide-State Circuits*, 30 :762–772, july 1995.

[MW01] M.Lai and D.F. Wong. Slicing tree is a complete floorplan representation. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'01)*, pages 228–232. DATE-01, 2001.

[Nag75] L.W Nagel. SPICE2 : A Computer program to simulate circuits. Technical Report ERL-M520, ERL, University of California, Berkeley, 1975.

[Nai99] Ravindranath Naiknaware. Automated Hierarchical CMOS Circuit Stack Generation with Intramodule Connectivity and Matching Considerations. In *IEEE Journal of the Solid States Circuit*, volume 34, pages 304–317. IEEE, IEEE, march 1999.

[ODJ+95] B. R. Owen, R. Duncan, S. Jantzi, C. Ouslis, S. Rezania, and K. Martin. BALLISTIC : An Analog Layout Language. *IEEE Custom Integrated Circuits Conference*, pages 3.5.1–3.5.4, 1995.

[OM95] B. R. Owen and K. Martin. BALLISTIC : An analog Layout Language. *Annual International Mentor Graphics User's Group (MUG) Conference*, 1995.

[OSC99] OSCI. SystemC community. http ://www.systemc.org, 1999.

[PDW89] M. J. M. Pelgrom, A.C. J. Duinmaijer, and A. P. G. Welbers. Matching properties of MOS transistors. In *IEEE Journal of the Solid States Circuit*, volume SC-24, pages 1433–1439. IEEE, IEEE, 1989.

[PKR+99] R. Phelps, M. Krasnicki, R. Rutenbar, R.L. Carley, and J. Hellums. ANACONDA : Robust synthesis of analog circuits via stochastic patern search. In

Proc. IEEE Custom Integrated Circuit Conference, pages 26.2.1–26.2.4, may 1999.

[Por05] J. Porte. Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiquEs. http ://www-asim.lip6.fr, 2005.

[RB94] W.R. Runyan and K.E. Bean. *Semiconductors Integrated Circuit Processing Technology*. Addison-Wesley, 1994.

[Sic02] E. Sicard. Microwind. http ://www.microwind.org, 2002.

[SSKN04] Gagandeep S. Sandha, Pawan K. Singh, C. Pradeep Kumar, and D. Nagchoudhuri. Quantitative Model for Thermal Behaviour of an Analog Integrated Circuit. In *17th International Conference on VLSI Design*, page 623. vlsid, 2004.

[STK84] J.B. Shuy, G.C. Temes, and F. Krummenacher. Random Error Effect in Matched MOS Capacitors and Current Sources. In *IEEE J. Solid-State Circuits*, volume SC-19, pages 948–956. IEEE, 1984.

[STY82] J.B. Shuy, G.C. Temes, and K. Yao. Random Error in MOS Capacitors. In *IEEE J. Solid-State Circuits*, volume SC-17, pages 1070–1076. IEEE, 1982.

[TLG04a] Pierre Nguyen Tuong, Marie-Minerve Louërat, and Alain Greiner. Guidelines for Designing Smart and Reusable Analog IP Cores. In *Sophia Antipolis MicroElectronics Forum (SAME 2004)*. SAME, october 2004.

[TLG04b] Pierre Nguyen Tuong, Marie-Minerve Louërat, and Alain Greiner. Managing the Shape Function of Analog Devices in a Slicing Tree Floorplan. In *Proceedings of the 11th International Conference Mixed Design of Integrated Circuits and Systems*, pages 226–229, Szczecin, Poland, june 2004. MIXDES 2004.

[Tsi99] Yannis Tsividis. *Operation and Modeling of the MOS Transistor*. Mac Graw-Hill, 1999.

[VdPS+01] Vancoreland, Van de Plas, Steyaert, Gielen, and Sansen. A layout-aware synthesis methodology for RF circuits. In *ICCAD*, pages 358–362. ICCAD 2001, november 2001.