Connaissance et synthèse en vue de la conception et la réutilisation de circuits analogiques intégrés

Ramy Iskander

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

Spécialité : **INFORMATIQUE ET MICRO-ÉLECTRONIQUE**

Présentée par : **Ramy ISKANDER**

Pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ PARIS VI

CONNAISSANCE ET SYNTHÈSE EN VUE DE LA CONCEPTION ET LA RÉUTILISATION DE CIRCUITS ANALOGIQUES INTÉGRÉS.

Soutenue le : **2 Juillet 2008**

Devant le jury composé de :

M.	Georges GIELEN	KUL, Rapporteur
M.	Maher KAYAL	EPFL, Rapporteur
M.	Andreas KAISER	IEMN-ISEN
Mme.	Noëlle LEWIS	IMS
M.	Hani RAGAI	UFE
M.	Serge SCOTTI	STMicroelectronics
M.	Alain GREINER	LIP6
Mme.	Marie-Minerve LOUËRAT	LIP6

Ph.D. THESIS OF THE UNIVERSITY OF PARIS VI

Department : **COMPUTER SCIENCE AND MICRO-ELECTRONICS**

Presented by : **Ramy ISKANDER**

Thesis submitted to obtain the degree of
DOCTOR OF THE UNIVERSITY OF PARIS VI

KNOWLEDGE-AWARE SYNTHESIS FOR ANALOG INTEGRATED CIRCUIT DESIGN AND REUSE.

Defended in 2nd July 2008

Committee in charge :

M.	Georges GIELEN	KUL, Reviewer
M.	Maher KAYAL	EPFL, Reviewer
M.	Andreas KAISER	IEMN-ISEN
Mme.	Noëlle LEWIS	IMS
M.	Hani RAGAI	UFE
M.	Serge SCOTTI	STMicroelectronics
M.	Alain GREINER	LIP6
Mme.	Marie-Minerve LOUËRAT	LIP6

A la mémoire de mon père et de mon beau-père.

Résumé

L'industrie des semi-conducteurs continue ses progrès impressionnants dans la miniaturisation des circuits intégrés VLSI. Les concepteurs ont inventé des méthodes permettant d'exploiter la complexité croissante des circuits intégrés à haute densité d'intégration. L'une d'elles consiste à concevoir des systèmes embarqués sur puce (SoC) à l'aide de blocs pré-existants et déjà validés (appelés IP, comme Intellectual Property), qu'ils aient été élaborés en interne à l'entreprise réalisant l'intégration du SoC ou issus d'une tierce partie. Disposer d'une bibliothèque de blocs IP paramétrés selon leurs performances en temps, surface et consommation est une clef pour optimiser le système intégré vis à vis de l'application ciblée. S'il existe un flot standard bien établi pour concevoir les blocs intégrés numériques, reposant sur une méthode de conception descendante, la conception de circuits analogiques reste toujours une opération sur mesure. Alors que les systèmes intégrés sur puce sont souvent mixtes analogique-numérique, les méthodes de conception diffèrent complètement entre les deux mondes.

Dans cette thèse, nous proposons une méthode pour automatiser le dimensionnement et la polarisation d'un circuit analogique dans le cas général, conduisant ainsi à une définition possible d'un IP analogique. Cette méthode permet de générer automatiquement une procédure pour calculer les dimensions d'une topologie électrique connue et son point de fonctionnement en se fondant sur l'expression de la connaissance du concepteur. Cette méthode permet de détecter des hypothèses conflictuelles émises par le concepteur et de traiter les cycles résultant des boucles de contre-réaction. Plusieurs circuits analogiques sont présentés pour illustrer la généralité et la précision de cette approche.

Mots Clefs

IP analogique, circuits analogiques intégrés CMOS, synthèse hiérarchique basée sur la connaissance, point de polarisation, analyse des dépendances, détection des conflits, résolution de conflits.

Abstract

The semiconductor industry has continued to make impressive improvements in the achievable density of very large-scale integrated (VLSI) circuits. In order to keep pace with the levels of integration available, design engineers have developed new methodologies and techniques to manage the increased complexity inherent in these large chips. One such emerging methodology is system-on-chip (SoC) design, wherein predesigned and preverified blocks (often called intellectual property (IP) blocks) are obtained from internal sources, or third parties and combined on a single chip. A library of reusable IP blocks with various timing, area, power configurations is the key to SoC success as the SoC integrator can apply the trade-offs that best suit the needs of the target application. Digital design has a well-defined, top-down design methodology but analog/mixed-signal (AMS) design has traditionally been an ad hoc custom design process. When analog and digital blocks coexist on the same substrate, the analog portion can be more time-consuming to develop even though it may represent a smaller percentage of the chip area.

In this thesis, we present a hierarchical sizing and biasing methodology for analog intellectual properties. The proposed methodology addresses the problem of automatically generating suitable designs plans that are used to compute the DC operating point and dimensions for analog IPs. The methodology deals with different aspects of analog design problems such as insufficient degrees of freedom, systematic offset and negative feedback circuits. It has been used to successfully size and bias a variety of analog IPs and proved its precision and efficiency.

Keywords

Analog Design Reuse, Hierarchical knowledge-based synthesis, Hierarchical sizing and biasing, DC analysis, Dependency analysis, Conflict detection, Conflict resolution.

Contents

List of Figures

List of Tables

Résumé Étendu en Français

Ce chapitre est un résumé étendu de la thèse, en français. Le lecteur intéressé par plus de détails pourra commencer directement la lecture au chapitre suivant.

1. Introduction

Ce paragraphe résume l'introduction de la thèse qui est l'introduction.

Voici plusieurs années que la synthèse de circuits numériques a atteint un niveau de maturité élevé. Le comportement de ces circuits peut être représenté à différents niveaux d'abstraction, allant des algorithmes jusqu'au niveau transistor, en utilisant des langages description standardisés (i.e. VHDL, Verilog et SystemC) et des bibliothèques de portes logiques, dites cellules standard. En ce qui concerne les circuits intégrés analogiques, la situation est bien différente. A chaque nouvelle application, les concepteurs doivent inventer un nouveau dimensionnement d'une topologie qu'ils auront sélectionnée, pour réaliser la fonction et les performances souhaitées. Il n'existe pas, aujourd'hui, de méthode ayant conduit à un consensus permettant de synthétiser automatiquement un circuit analogique dans le cas général. Les outils de conception couramment utilisés sont les simulateurs fonctionnels (type MATLAB), les simulateurs au niveau transistor (type SPICE) et les éditeurs de masques. Par ailleurs, si un circuit analogique a donné satisfaction dans le cadre d'une application (SoC), il est difficile de le réutiliser dans un autre contexte, sans dégrader les performances de la nouvelle application.

Cette thèse est une contribution à la synthèse des circuits intégrés analogiques. Elle propose une méthode générale pour calculer les dimensions et la polarisation des transistors d'un circuit analogique CMOS en se fondant sur la connaissance du concepteur, sans avoir recours à un simulateur. Cette méthode suppose que le circuit est décrit comme une hiérarchie de modules et de dispositifs élémentaires dans l'environnement de conception CAIRO+. Un dispositif élémentaire est composé d'un petit nombre de transistors interconnectés, parmi lesquels on définit un transistor de référence et des transistors secondaires. Ce transistor de référence a un rôle essentiel puisqu'il contrôle le dimensionnement et la polarisation des autres transistors au sein du dispositif élémentaire. Ce contrôle est exprimé sous la forme d'un graphe de dépendance. A partir de la description hiérarchique du circuit total, on peut construire son graphe de dépendance. Celui-ci permet d'assurer que les contraintes électriques sont satisfaites par construction et exprime de

dimensionnement du circuit résultant des hypothèses du concepteur.

Cependant, à cause du nombre élevé de degrés de liberté présents dans un circuit analogique, des conflits peuvent apparaître dans ce graphe. La méthode proposée permet de détecter automatiquement les conflits. Dans certains cas, l'introduction d' un degré de liberté supplémentaire permet la résolution du conflit sous la forme d'une tension de décalage systématique.

Notre contribution porte sur quatre points :

- Pour atteindre la précision d'un simulateur au niveau transistor, nous avons introduit un mécanisme d'inversion du modèle électrique du transistor, mis en oeuvre dans une **bibliothèque d'opérateurs électriques**. Le modèle électrique utilisé pour le transistor est BSIM3v3.

- Nous avons conçu une méthode pour **dimensionner et polariser** un circuit analogique en nous appuyant sur un graphe de dépendance. Cette méthode suppose connue la connectique du circuit (topologie électrique) et effectue un changement de variables du problème de dimensionnement, ce qui permet de prendre en compte la connaissance du concepteur et de restreindre l'espace des solutions possibles. Nous exprimons le dimensionnement d'un circuit en termes d'opérateurs électriques.

- Pour expérimenter cette méthode, nous avons conçu un **langage et un moteur de dimensionnement** qui ont été intégrés à l'environnement de conception CAIRO+.

- Cette méthode a été appliquée avec succès à plusieurs circuits analogiques : un OTA 2 étages, un intégrateur gm/C en mode courant, un amplificateur différentiel avec stabilisation du mode commun, un amplificateur en transconductance et un amplificateur différentiel sous très basse tension d'alimentation.

2. Position du problème et Motivation

Ce paragraphe résume le chapitre 2 de la thèse qui présente le contexte de l'étude en définissant le problème à résoudre et en introduisant les objectifs du travail.

2.1. Le contexte : les systèmes intégrés sur puce

En numérique, avant le standard VHDL, les concepteurs décrivaient les circuits comme une interconnexion de portes logiques, au niveau structurel et physique. Avec le standard VHDL, les concepteurs ont pu décrire les circuits sous forme d'automates. Ce sont les outils de synthèse logique, basés sur ces automates, qui construisent la netlist en porte. Celle-ci est utilisée par des outils de placement et routage pour dessiner les masques du circuit.

Ces résultats associés à l'augmentation de la finesse de gravure sur silicium, ont permis d'envisager la conception d'un système intégré sur puce (dit SoC) à base de composants, appelés

IP (Intellectual Property), décrits par leur vue comportementale. Ce sont aux outils de synthèse et de placement et routage que revient la charge de dessiner les masques en s'appuyant sur une bibliothèque de cellules standard.

En analogique, la situation est tout autre. S'il existe aujourd'hui des langages tels que VHDL-AMS ou SystemC-AMS permettant de modéliser la fonction souhaitée, il n'existe pas d'algorithme et encore moins d'outil logiciel permettant de synthétiser la fonction dans le cas général. Un consensus se dégage pourtant pour définir ce que pourrait être un IP analogique, dit aussi *Firm IP*. Il ne s'agit ni d'une description du matériel seul (le dessin de masques entièrement figé), ni du logiciel seul (description fonctionnelle facilement reconfigurable), mais d'un triplet constitué d'une netlist définissant une structure électrique bien identifiée, un ensemble de paramètres pour dimensionner cette structure et un plan de masse approximatif (placement relatif de composants dont on ne connaît pas encore les dimensions). Le flot de conception comporte alors deux phases essentielles : le dimensionnement de la structure et le dessin des masques.

Dans cet esprit, CAIRO+ définit un IP analogique comme un circuit paramétrable à base de dispositifs élémentaires. Le concepteur a la charge de décrire la structure, les paramètres et un plan de masse relatif. CAIRO+ dispose d'un moteur de dimensionnement des dispositifs élémentaires et de dessin des masques. Ces moteurs sont paramétrables en fonction de la technologie. L'exécution du code avec des valeurs de spécifications fournit les dimensions de la structure, les performances estimées et le dessin des masques. Concevoir un circuit paramétrable reste une lourde tâche qui n'a d'intérêt que pour les blocs réutilisables, suivant d'autres spécifications ou d'autres technologies.

L'objectif de cette thèse est de décharger le concepteur d'une partie de la tâche de dimensionnement en lui proposant une méthode générale et des outils pour l'expérimenter, permettant de déduire automatiquement une procédure de dimensionnement, quelle que soit la structure électrique cible. On se propose pour cela de formaliser le processus de **dimensionnement hiérarchique** suivant le calcul du point de polarisation.

2.2. Approche classique pour dimensionner un circuit analogique

Considérons le filtre passe-bas de la figure 1 où l'amplificateur est réalisé par l'OTA simple de la figure 2 comme exemple pour analyser le raisonnement du concepteur lors du dimensionnement (Fig. 3), et en déduire une méthode de dimensionnement hiérarchique basée sur le calcul du point de polarisation.

Suivant l'application visée, les spécifications portant sur le filtre vont imposer des bornes minimales au gain A_{d0} et à la fréquence de transition F_T de l'OTA (Fig. 3). En s'appuyant sur le modèle simple du transistor MOS, ces valeurs permettent d'en déduire le courant I_{DS,M_1} et les longueurs minimales des transistors L. Les tensions de mode commun V_{OUT}, V_{IN+} et V_{IN-} sont imposées par les circuits d'entrée et de sortie de l'OTA. Le concepteur peut choisir de spécifier la tension effective de grille $V_{eg} = V_{GS} - V_{th}$ pour tous les transistors [Porte08, Silveira96, Stefanovic03,

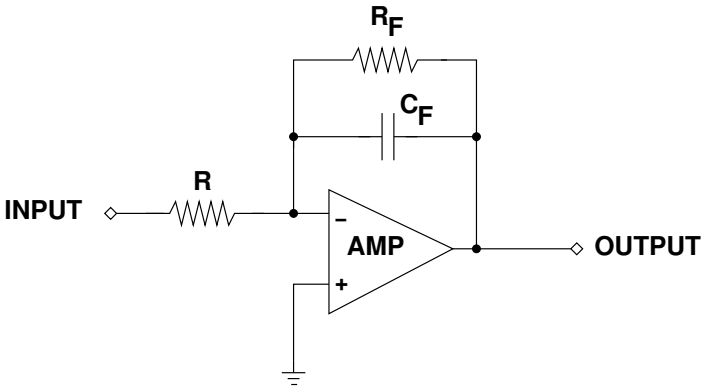

Figure 1: *Filtre passe-bas.*

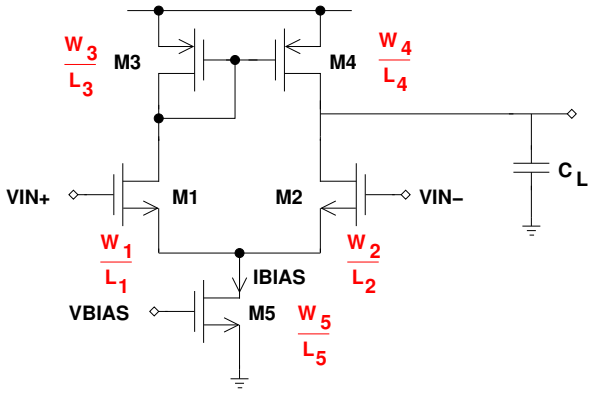

Figure 2: *Amplificateur opérationnel à transconductance (dit OTA simple).*

Binkley03]. L_1, I_{DS1}, V_{DS1}, μ_n, C_{ox} et λ_n étant alors connus, la largeur de M_1 peut être déduite de l'équation 1:

$$I_{DS}(NMOS) = \frac{\mu_n}{2} C_{ox} \frac{W}{L} (V_{eg})^2 (1 + \lambda_n V_{DS}) \tag{1}$$

Ensuite le concepteur utilise l'équation 2 pour déterminer W_3:

$$I_{DS}(PMOS) = \frac{\mu_p}{2} C_{ox} \frac{W}{L} (V_{eg})^2 (1 + \lambda_p V_{DS}) \tag{2}$$

Une fois M_1 et M_3 dimensionnés, on copie leurs dimensions pour M_2 et M_4. M_5 est ensuite dimensionné, en utilisant $V_{BIAS} = V_{eg5} + V_{th5}$ et l'équation 1 pour calculer W_5. Une fois que toutes les dimensions électriques et physiques (W et L) sont connues, on est en mesure de calculer les paramètres petits-signaux et les performances électriques qui en découlent (Fig. 3).

2.3. Approche proposée

Nous nous proposons de généraliser ce raisonnement. Si le problème de dimensionnement d'un circuit peut être posé en termes de calcul des largeurs et longueurs de tous les transistors, il

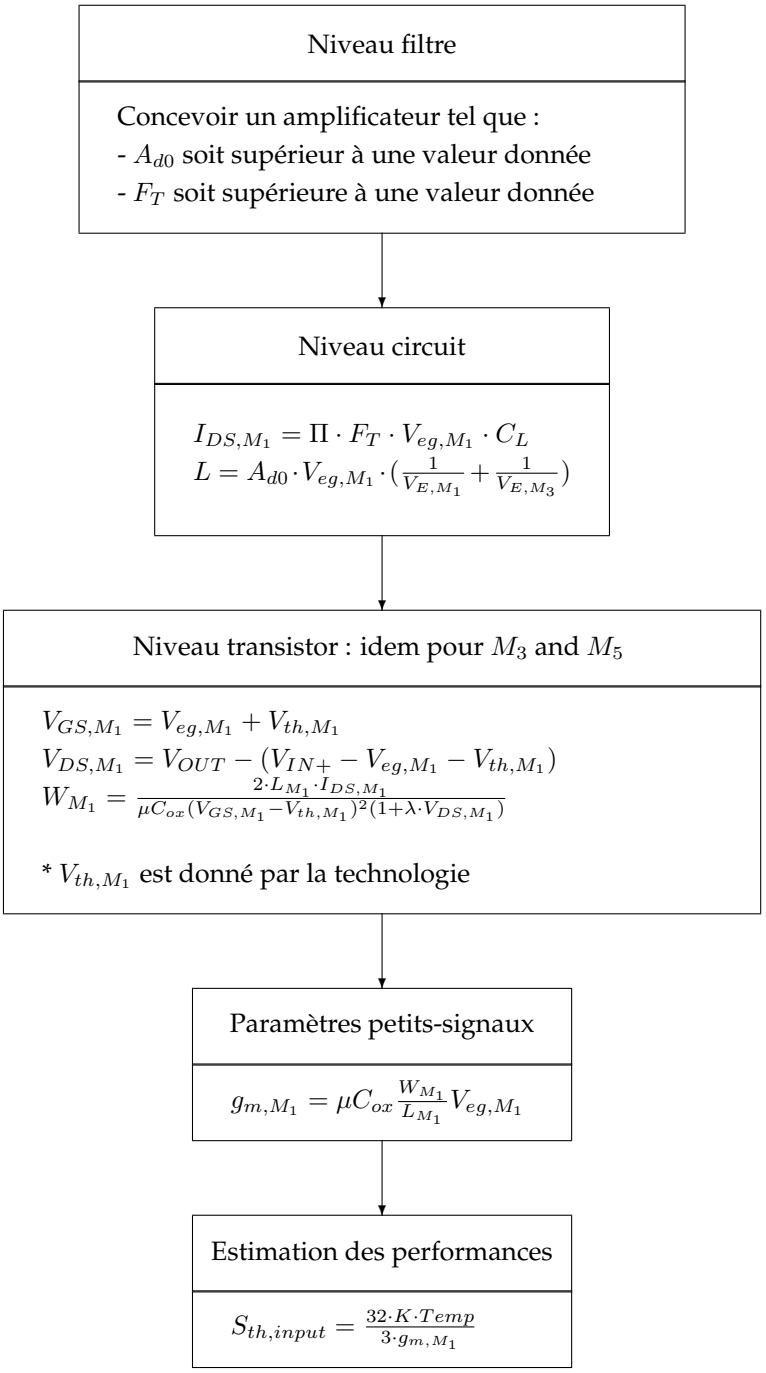

Figure 3: *Procédure de dimensionnement descendante de l'OTA simple (Fig. 2).*

apparaît qu'un changement de variable en termes de courant de polarisation et tension effective de grille (Fig. 4) rend le problème plus facile à formuler par le concepteur. Cette idée de la synthèse basée sur la polarisation est utilisée par plusieurs autres approches [Silveira96, Leyn98, Stefanovic07, Porte08, Binkley03]. Une fois connu le point de polarisation, on peut calculer les largeurs de tous les transistors puis leurs paramètres petits signaux.

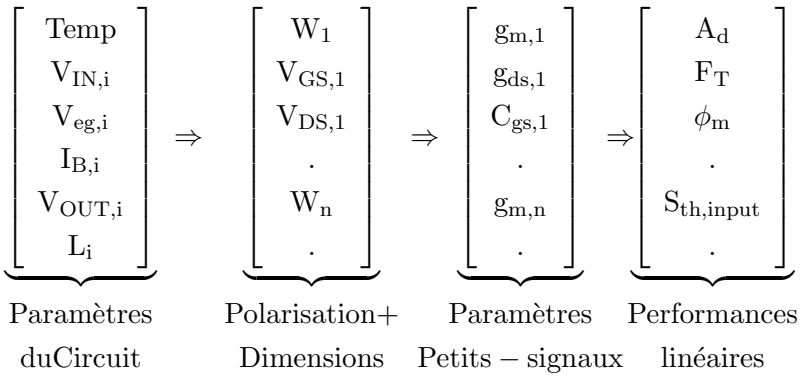

Figure 4: *Changement de variables et paramètres de dimensionnement*.

Ce graphe permet d'examiner la cohérence des hypothèses effectuées sur le circuit et, dans le cas de graphes sur-contraints ou sous-contraints, de proposer un diagnostic pour résoudre le conflit.

2.4. La synthèse d'un circuit analogique

L'idée directrice de notre méthode est de montrer les bénéfices apportés par l'utilisation conjointe de la connaissance et de l'optimisation dans un problème de synthèse analogique. Habituellement, l'optimisation utilise les largeurs des transistors comme variables à optimiser de façon à évaluer les performances du circuit à l'aide d'un simulateur électrique au niveau transistors. Or, considérer les largeurs des transistors comme variables n'est en général pas bien adapté au problème. En effet, ce choix conduit à un espace de conception très vaste du fait de l'étendue des valeurs possibles pour ces largeurs. La procédure de synthèse peut alors passer un temps considérable à évaluer des solutions non réalisables physiquement puisque largeurs et longueurs des transistors sont choisies indépendamment l'une de l'autre.

Un autre point de vue consiste à considérer le premier vecteur donné dans la figure 4 comme ensemble de variables à optimiser. Ce vecteur contient des courants, des tensions de polarisation et des longueurs de transistors. Ces variables sont définies sur un intervalle plus petit que celui des largeurs. En conséquence, pour un circuit donné, les variables du vecteur 1 définissent un espace de conception plus petit que celui issu des largeurs. De plus, comme les largeurs sont calculées à partir de ces variables, les dimensions des transistors sont cohérentes avec les hypothèses de polarisation. La méthode de dimensionnement automatique des transistors que nous avons

développée permet ensuite de calculer les composantes du deuxième vecteur de la figure 4 à partir des composantes du premier vecteur.

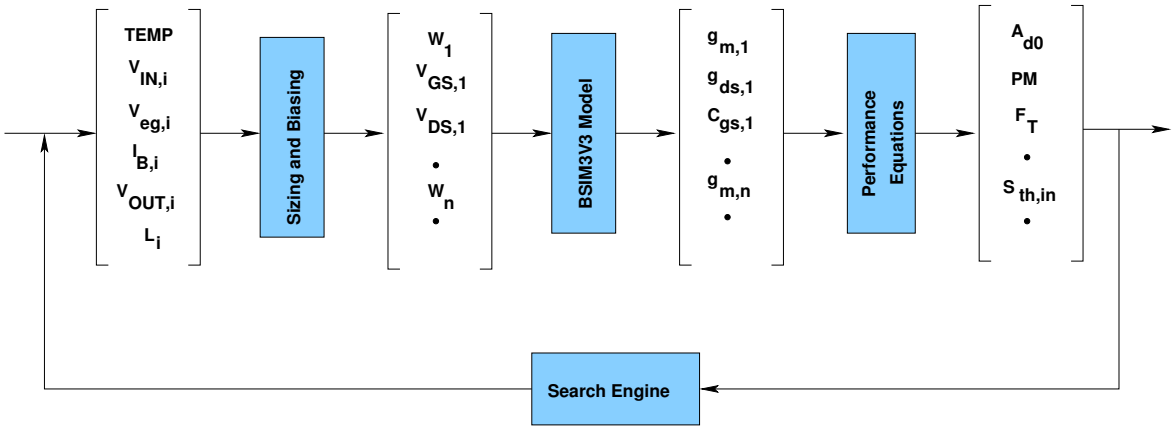

Figure 5: *Position du problème de synthèse d'un circuit analogique* .

Cette approche (Fig. 5), combinant optimisation et connaissance, permet d'accélerer la synthèse de circuits analogiques, tout en garantissant une précision comparable à celle d'un simulateur.

3. Etat de l'art

Ce paragraphe résume le chapitre 3 de la thèse qui propose un état de l'art du domaine.

3.1. Polarisation et dimensionnement

Depuis une vingtaine d'années, les études portant sur la synthèse analogique ont été partagées entre deux "écoles" : l'une est fondée sur la connaissance et l'autre sur la simulation électrique. Dans le cas où la synthèse est fondée sur la connaissance, c'est au concepteur que revient la charge de capitaliser son savoir faire sous une forme qui puisse être réutilisée. Un des premiers outils à suivre cette approche est OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a] qui réalise la capitalisation du savoir faire via le codage du calcul des dimensions et du point de polarisation (point DC) de chacun des sous-circuits qui composent le circuit. Cette phase de codage est jugée très fastidieuse par la plupart des concepteurs. Inversement, dans le cas où la synthèse est réalisée à partir de simulations électriques, on doit appeler un simulateur pour chaque point DC à résoudre. Cette approche conduit à un temps d'exécution total conséquent, comme cela apparaît dans MAELSTROM [Krasnicki99] dont la solution, mettant en oeuvre une optimisation, nécessite plusieurs centaines de simulation. Notons que, dans les deux approches, le calcul du point de polarisation DC reste une étape incontournable.
C'est pourquoi plusieurs auteurs se sont intéressés au calcul du point DC et ont apporté des

solutions à ce problème fondamental. Maulik [Maulik91, Maulik92b, Maulik92a] a utilisé une méthode de relaxation en introduisant la solution DC comme un terme d'une fonction de coût. Gielen *et al* [Plas01] ont résolu le problème du point DC comme un problème d'optimisation, dont les variables sont les courants et les tensions de branche. Les dimensions des transistors sont ensuite déduites en inversant le modèle électrique. L'inconvénient de ces méthodes est de faire appel aux techniques d'optimisation, coûteuses en temps d'exécution, pour résoudre le point DC.

3.2 Connaissance et synthèse

Plusieurs études, dont le but était d'automatiser le dimensionnement d'un circuit analogique, ont cherché à exprimer la connaissance sous forme d'une succession d'étapes de conception. Swings et Sansen ont proposé DONALD [Swings91c] pour inverser numériquement le modèle analytique comportemental d'un circuit analogique. DONALD utilise un algorithme numérique qui permet d'inverser les équations en fonction de différents paramètres, correspondant aux variables du modèle. Ceci permet d'utiliser le même modèle pour résoudre des problèmes de synthèse ou d'analyse. Le modèle de conception est représenté sous forme d'un graphe bipartite non orienté. Il est transformé en une séquence ordonnée d'équations de dimensionnement, appelée *solution plan* à partir de la propagation des contraintes, représenté par un graphe bipartite orienté. Ce graphe est utilisé dans un processus de synthèse pour calculer les dimensions de composants élémentaires à partir de performances spécifiées. DONALD sait traiter les problèmes sur-contraints ou sous-contraints. Les cas sous-contraints correspondent à une donnée manquante. Les cas sur-contraints correspondent à des variables qui doivent satisfaire plusieurs équations. DONALD détecte ces problèmes et propose des solutions au concepteur. Bernardinis et Sangiovanni [Bernardinis04], eux, ont formalisé le problème de dimensionnement d'un circuit analogique par un graphe bipartite appelé *Analog Constraint Graphs* (ACGs). L'espace de conception admissible est défini par l'ensemble des contraintes d'égalité et d'inégalité portant sur les variables de conception. Cette représentation est utilisée pour réduire l'espace de conception. Les ACGs permettent de choisir efficacement, dans l'espace de conception, des jeux de variables correspondant uniquement aux solutions réalisables. C'est pourquoi les ACGs sont utilisés lors de l'exploration du domaine de conception, pour modéliser les performances du circuit.

3.3. Comparaison des outils de synthèse

Le tableau à la fin de ce chapitre présente un comparatif des outils de synthèse.

3.4 L'environnement de conception Cairo

Cette thèse a contribué à l'environnement de conception CAIRO+ développé depuis une dizaine d'années au laboratoire LIP6 dans le cadre des thèses de Mohamed Dessouky, Hassan

Aboushady, Pierre Nguyen Tuong, Vincent Bourguet, Nicolas Beilleau, Jose Bonan et Laurent de Lamarre [Dessouky01, Tuong06, de Lamarre02, Iskander04]. CAIRO+ permet de concevoir des générateurs paramétrables. Un générateur paramétrable est un composant réutilisable (fig. 6) qui reçoit en entrée les paramètres électriques et physiques du procédé de fabrication, ainsi que des valeurs de spécifications et qui fournit en sortie une liste de performances, une netlist dimensionnée et le dessin des masques correspondant.

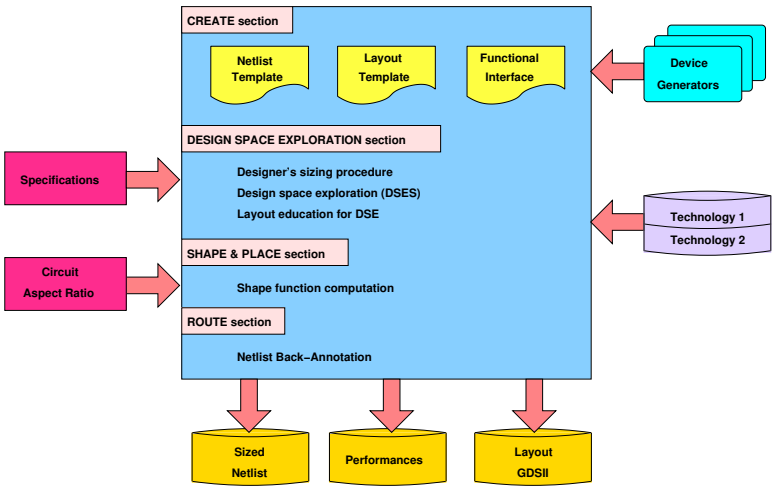

Figure 6: *Architecture logicielle d'un générateur CAIRO+ .*

Les dispositifs élémentaires et les modules sont des générateurs paramétrables. Un circuit analogique peut être représenté par une hiérarchie de modules et de dispositifs élémentaires. Les modules peuvent instantier des modules existants et des dispositifs élémentaires. Le niveau le plus bas de la hiérarchie est le niveau transistor. Chaque niveau de la hiérarchie ne peut communiquer qu'avec le niveau directement supérieur ou avec ses descendants directs. La figure 7 présente un exemple de description et de communication hiérarchique avec CAIRO+.

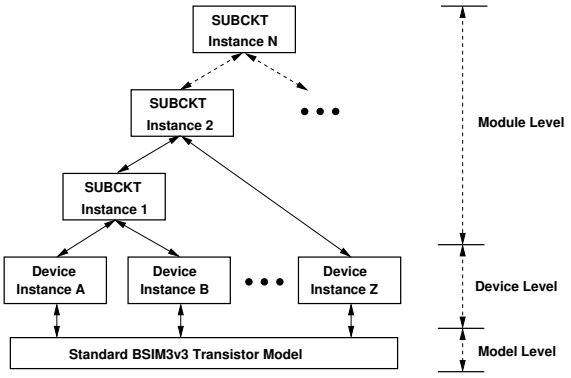

Figure 7: *Niveaux hiérarchiques et propagation des paramètres avec CAIRO+.*

Table 1: *Opérateurs utilisés pour le dimensionnement*.

Opérateur	Définition
$OPVS(V_{eg}, V_B)$	$(V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G, V_B$
$OPVS(V_{eg})$	$(V_S, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G$
$OPVG(V_{eg})$	$(V_B, V_G, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S$
$OPVGD(V_{eg})$	$(V_B, V_G, V_D, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S$
$OPW(V_G, V_S)$	$(V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_D, V_G, V_S$

4. Polarisation et dimensionnement d'un transistor

Ce paragraphe résume le chapitre 4 de la thèse qui présente un modèle inverse du transistor en vue de son dimensionnement.

Lorsqu'ils suivent l'approche courante, les concepteurs utilisent le modèle quadratique(équations 1 et 2) pour calculer "manuellement" les largeurs des transistors. Cette équation est un modèle très simplifié du transistor MOS. Comme un de nos objectifs est d'atteindre la précision d'un simulateur, nous avons introduit 46 opérateurs de dimensionnement dans CAIRO+. Chaque opérateur se présente sous la forme suivante :

$$OP<class>(RV_i, ...) : (LV_j, ...) \Leftarrow (RV_n, ...) \tag{3}$$

où $<class>$ indique le paramètre essentiel à calculer, RV_i est un sous ensemble de paramètres connus du concepteur qui indique la version de l'opérateur, RV_n est l'ensemble des paramètres d'entrée nécessaires pour exécuter l'opérateur et LV_j est l'ensemble des paramètres inconnus qui sont calculés par cet opérateur. Un paramètre est considéré "connu" s'il est fixé par le concepteur ou s'il résulte d'un calcul précédent de CAIRO+. Le tableau 1 présente les opérateurs utilisés pour le dimensionnement. A titre d'exmple, considérons l'opérateur $OPVS$. Sa classe est celle de la *tension de source*. Cet opérateur a deux versions suivant la connexion du "bulk". La première $OPVS(V_{eg}, V_B)$ est activée dans le cas où la tension V_{eg} est connue et que la tension V_B doit être fixée. Elle calcule V_S, V_{th} et W, simultanément, en fonction des paramètres apparaissant à droite de la flèche dans le tableau 1. La seconde version $OPVS(V_{eg})$ est activée si V_{eg} est connue et que les connecteurs "bulk" et source sont reliés à l'intérieur du dispositif.

Dans l'environnement CAIRO+, le concepteur a ainsi à sa disposition plusieurs fonctions qui lui permettent d'inverser un modèle électrique précis (identique à un simulateur) en fonction des paramètres connus [de Lamarre02, Iskander08]. Ces valeurs peuvent être connues du concepteur ou bien résulter de la connectique du circuit. La bibliothèque d'opérateurs a été définie en examinant les divers cas de connexion illustrés par la figure 8.

Dans cette approche, la tension de drain peut être soit connue a priori, soit déterminée par sa connexion à un autre dispositif élémentaire par la grille ou la source (Fig. 8). Ces fonctions sont illustrées par la figure (9).

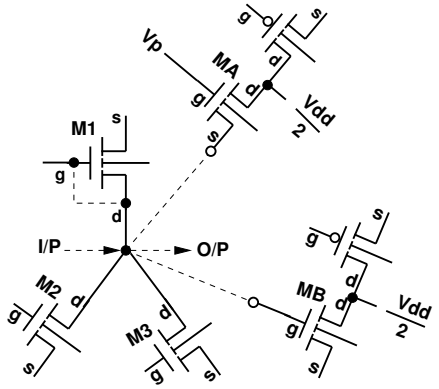

Figure 8: *Les connexions possibles d'un drain de transistor MOS.*

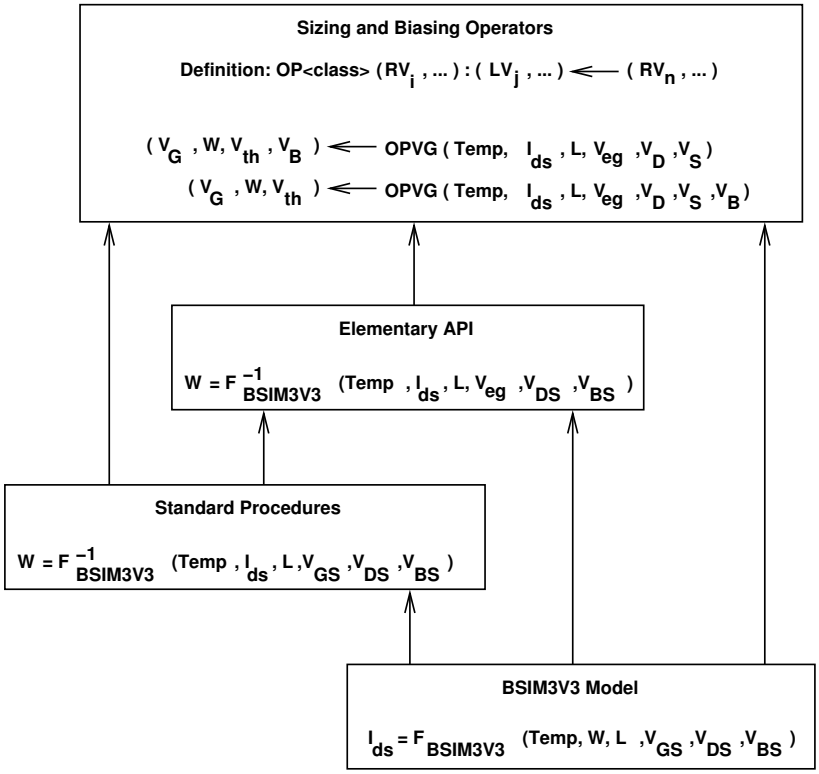

Figure 9: *Modèle électrique du MOS et différentes inversions possibles.*

5. Polarisation et dimensionnement d'un dispositif élémentaire

Ce paragraphe résume le chapitre 5 de la thèse qui introduit le concept de dispositif élémentaire et de transistor de référence pour structurer un circuit.

5.1. Définition d'un dispositif élémentaire

Un *dispositif élémentaire* est un groupe de transistors, réunis pour former une primitive réutilisable. Il constitue une feuille de la représentation hiérarchique d'un circuit analogique.

La figure 2 montre que l'OTA est constitué par un ensemble de 3 primitives : le miroir de courant (M_3, M_4), la paire différentielle (M_1, M_2) et le transistor MOS M_5. Afin de définir un dispositif élémentaire, on a identifié des règles pour regrouper certains transistors. Les conditions qui suivent doivent être satisfaites pour réunir des transistors au sein d'un unique dispositif élémentaire. Il s'agit de :

1. Les transistors qui constituent une "fonction" analogique.

2. Les transistors doivent être appariés (car ils ont des paramètres électriques communs).

On voit dans la figure 2 que les transistors M_3 and M_4 doivent être dans le même dispositif car :

1. Ils réalisent la fonction "miroir de courant".

2. Leurs paramètres W, L et V_{GS} devant être identiques, ils doivent être appariés.

Ces mêmes conditions sont remplies dans le cas de la paire différentielle (M_1, M_2).

La figure 10 présente la famille de dispositifs élémentaires disponibles dans l'environnement CAIRO+. Le dessin des masques a été réalisé dans la thèse de Vincent Bourguet [Bourguet04].

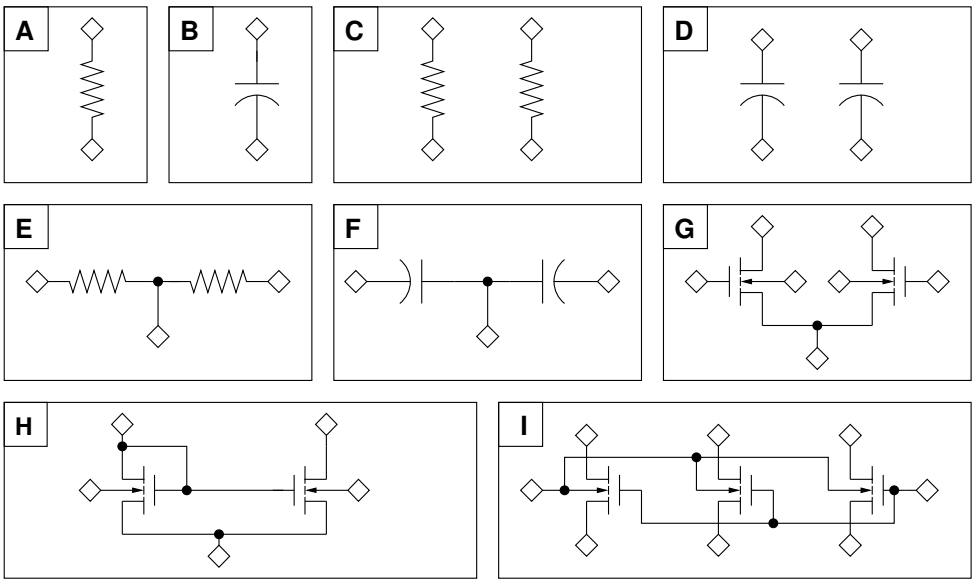

Figure 10: *La bibliothèque de dispositifs élémentaire.*

5.2. Le transistor de référence

Dans un souci d'abstraction, le concepteur choisit de dimensionner un nombre minimum de transistors. Pour définir ces transistors, on introduit le concept de *transistor de référence*. Un dispositif élémentaire contient un seul transistor de référence qui est dimensionné en premier. Les paramètres électriques de ce transistor définissent, d'une manière unique, par des relations simples, ceux des autres transistors. On peut dire que les paramètres de ce transistor sont *propagés* vers les transistors secondaires. Dans le cas de l'OTA de la figure 2, le concepteur peut choisir :

1. M_1 comme transistor de référence pour (M_1, M_2)
2. M_3 comme transistor de référence pour (M_3, M_4)
3. M_5 comme transistor de référence pour lui même.

Le transistor de référence de la paire différentielle est marqué par un point sur la figure 11.

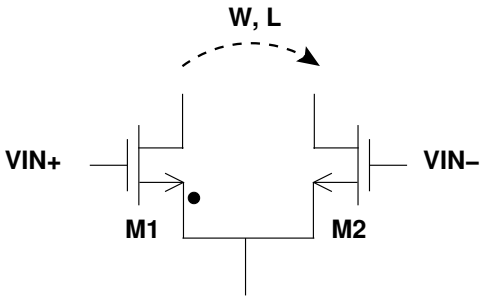

Figure 11: *Transistor de référence et propagation des paramètres d'une paire différentielle.*

5.3. Le graphe de dépendance d'un dispositif élémentaire

Dans l'approche classique décrite dans le cas du dimensionnement de l'OTA simple (Fig. 2), le concepteur a reporté les largeurs de M_1 et M_3 vers M_2 et M_4. Ceci signifie que certains paramètres doivent être propagés depuis le transistor de référence vers les transistors secondaires, au sein d'un même dispositif élémentaire. Dans le cas général on dénit les contraintes linéaires sous la forme :

$$\left[P_{elec,i}\right]_{N\times 1} = \left[K_i\right]_{N\times M} \cdot \left[P_{elec,ref}\right]_{M\times 1} \tag{4}$$

où $P_{elec,i}$ est la matrice dont les éléments sont les paramètres électriques des transistors secondaires, K_i est une matrice creuse, dont les éléments sont des constantes et $P_{elec,ref}$ est la matrice dont les éléments sont les paramètres électriques du transistor de référence. On peut définir plusieurs types de contraintes dans un dispositif élémentaire en s'inspirant de *sizing rules method*[Graeb01].

L'idée est alors de déduire automatiquement une procédure de dimensionnement pour un circuit entier, en se fondant sur la description hiérarchique en module et en dispositifs élémentaires. Une procédure d'un module ou d'un dispositif élémentaire est décrite par un graphe de dépendance où un noeud est un paramètre de dimensionnement et l'existence d'un arc orienté exprime une relation de dépendance entre la destination et l'origine. Un noeud représente les paramètres électriques comme $Temp$, W, L, I_{DS}, V_{eg}, V_{GS}, V_D, V_B, V_S ou V_G. Les arcs représentent une relation pondérée par le poids de l'arc, entre les paramètres du noeud v et celui du noeud u. Ce qui s'écrit : $v \leftarrow u$. L'existence d'un arc entre 2 noeuds résulte soit :

1. de la connectique, soit
2. de l'existence d'une relation entre les deux paramètres au travers d'un opérateur
3. de l'existence d'une contrainte liant ces paramètres.

Ce graphe garantit donc, par construction, le respect des contraintes existant pour chacun des dispositifs élémentaires.

La figure 12 illustre un exemple de propagation des largeurs entre transistors d'un miroir de courant.

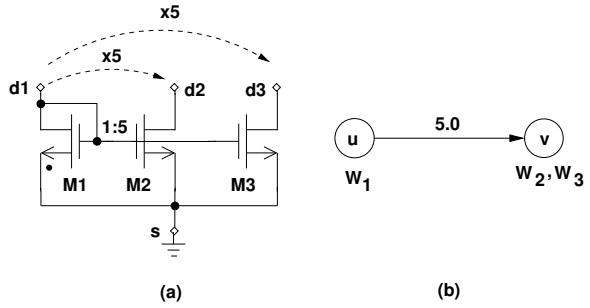

Figure 12: *Miroir de courant : (A) Contraintes sur les largeurs, (b) Propagation des valeurs.*

A partir de la structure du dispositif élémentaire, CAIRO+ détermine quel est l'opérateur adapté au transistor de référence. Le graphe de dépendance est construit en fonction de cet opérateur. Il représente la procédure de dimensionnement du dispositif. L'exemple de la figure 13 présente le graphe de dépendance qui a été généré suivant l'opérateur $OPVGD$ pour le miroir de courant (fig. 12). Ce graphe montre les relations de dépendance entre les paramètres inconnus V_S et W et les paramètres connus : $Temp$, I_{DS}, L, V_{eg}, V_D, V_B et V_G. Il est clair que chaque dispositif élémentaire possède un sous ensemble d'opérateurs en fonction de sa structure interne.

6. Polarisation et dimensionnement d'un circuit

Ce paragraphe est un résumé des chapitres 6 et 7 de la thèse. Le chapitre 6 présente le calcul automatique du point de polarisation d'un circuit fondé sur la structuration en modules et dis-

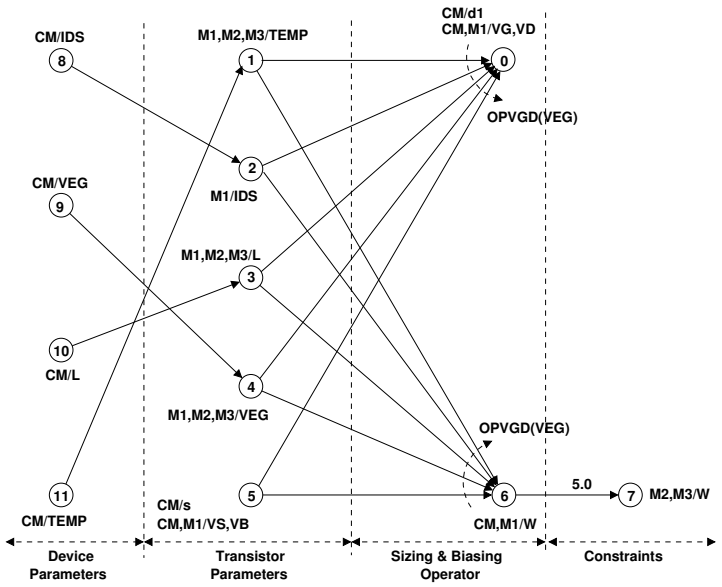

Figure 13: *Graphe de dépendance pour le miroir de courant (Fig. 12).*

positifs élémentaires. Le chapitre 7 met en oeuvre cette méthode pour dimensionner des circuits. Dans ce résumé on présente uniquement différents résultats obtenus pour l'OTA Miller.

6.1. Graphe de dépendance du circuit

Un circuit est représenté comme une hiérarchie de sous-circuits dans l'environnement CAIRO+. Les feuilles de la hiérarchie sont des *dispositifs élémentaires* et les niveaux supérieurs sont les *modules*. Chaque sous-circuit est représenté par son graphe de dépendance. Le graphe de dépendance exprime les relations qui existent entre les paramètres électriques DC (tensions, courants et dimensions) et certains paramètres choisis comme paramètres d'entrée. Au cours de la synthèse, on construit automatiquement le graphe de dépendance à partir du transistor de référence de chacun des dispositifs élémentaires. Ainsi le graphe du circuit entier est construit, récursivement, en commençant par les dispositifs élémentaires.

Ce graphe est transformé, si possible, en graphe orienté acyclique (DAG).

Lorsque le graphe résultant est un DAG il représente un plan de dimensionnement du circuit analogique. Pour dimensionner le circuit, le plan de dimensionnement est exécuté d'une manière descendante, pour calculer le point de polarisation et les largeurs de tous les transistors à partir des paramètres fixés par le concepteur.

6.2. Existence de conflits dans le graphe de dépendance

Du fait du nombre élevé de degrés de liberté d'un problème de synthèse analogique, il peut apparaître des conflits dans le graphe de dépendance. Les conflits apparaissent quand un même

paramètre est défini de plusieurs façons, i.e. il existe plusieurs chemins dans le graphe aboutissant à un même paramètre ou au contraire quand un ou plusieurs paramètres ne sont pas définis. Ces conflits rendent la connaissance introduite dans le graphe, incohérente. Nous allons montrer comment enrichir la connaissance du circuit qui repose sur l'existence de ce graphe pour détecter et supprimer les conflits et construire un graphe orienté acyclique. Il est essentiel de savoir identifier les divers cas de conflits, de les comprendre et de les résoudre pour modéliser de manière satisfaisante l'expertise du concepteur.

Nous avons examiné le problème particulier posé par l'introduction d'une tension de décalage systématique lors de la conception d'un amplificateur.

Nous avons montré que cette tension se manifeste par l'existence de plusieurs hypothèses conflictuelles. En identifiant le noeud sur lequel se produit le conflit, nous avons pu estimer précisément la valeur de la tension de décalage.

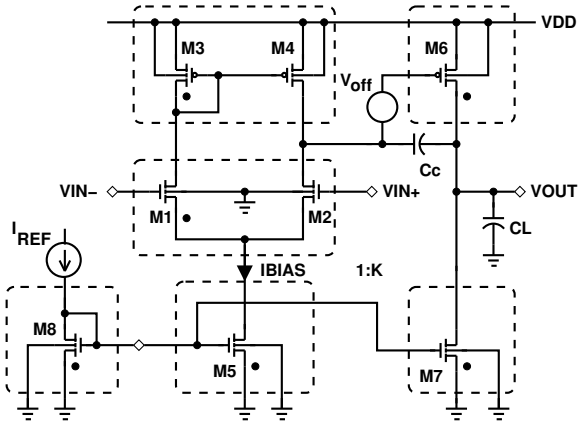

Figure 14: *L'OTA deux étages avec introduction d'un degré de liberté supplémentaire.*

6.2.1. Tension de décalage

Si les deux entrées d'un amplificateur différentiel sont reliées à la tension de mode commun, la sortie devrait atteindre le mode commun de sortie, ce qui n'est en général pas le cas dans les circuits fabriqués. On parle alors de tension de décalage systématique ou aléatoire. Le décalage systématique résulte directement de la technique suivie par le concepteur. Un décalage de tension peut être acceptable dans certains cas d'amplificateurs opérationnels, de comparateurs, de convertisseurs analogique-numérique et numérique-analogique ... . La tension aléatoire résulte des dispersions des paramètres caractérisant le procédé de fabrication, elle est inévitable. Pour amener la sortie du circuit au niveau du mode commun souhaité, il est souvent nécessaire d'ajouter une tension à une des entrées, appelée tension de décalage ramenée à l'entrée.

Étudions l'amplificateur 2 étages présenté Fig. 14. L'augmentation de la transconductance du deuxième étage $g_{m,M6}$ permet d'augmenter le produit gain-bande GBW (à marge de phase

constante et capacité de charge constante). Comme :

$$g_{m,M6} \approx \frac{2I_{M6}}{V_{GS,M6} - V_{th,M6}} \approx \frac{2I_{M6}}{V_{eg,M6}} \tag{5}$$

la tension $V_{GS,M6}$, et la tension effective de grille $V_{eg,M6}$, doivent être faibles. Cette hypothèse entre en contradiction avec le choix arbitraire de la tension $V_{DS,M4}$. Ce conflit déséquilibre l'amplificateur qui sature au niveau V_{DD}. Pour équilibrer l'amplificateur, on ajoute un degré de liberté en libérant la tension au noeud $V_{D,M4}$. La différence entre $V_{D,M4}$ et $V_{G,M6}$ est la tension de décalage qui apparaît à la sortie du premier étage. Pour "ramener" à l'entrée cette tension, on divise cette valeur par le gain du premier étage.

$$V_{i,off} \approx (V_{D,M4} - V_{G,M6}) \cdot \frac{g_{ds,M2} + g_{ds,M4}}{g_{m,M1}} \tag{6}$$

En généralisant ce principe, on peut résoudre les conflits de tension sur n'importe quel noeud du graphe en introduisant une tension de décalage.

6.2.2. Détection du conflit

Pendant la construction du graphe de dépendance, un conflit apparaît dans le cas où un même paramètre (ici, une tension) est évalué par des opérateurs distincts. Comme il n'y a aucune raison que les valeurs données par les opérateurs distincts soient identiques, un conflit apparaît (Fig. 15(a)). Supposons (M_3,M_4) idéal (i.e. $V_{G,M3} = V_{D,M4}$). $V_{G,M3}$ et $V_{D,M4}$ partagent le même noeud. L'opérateur *OPVGD* est utilisé pour calculer $V_{G,M3}$ et $V_{D,M4}$ à partir des données connues (courant $I_{DS,M3}$). L'opérateur *OPVG* est utilisé pour calculer $V_{G,M6}$ à partir de données connues de M_6 (courant $I_{DS,M6}$). $V_{D,M4}$ et $V_{G,M6}$ formant une équipotentielle, $V_{G,M6}$, $V_{G,M3}$, $V_{D,M4}$ et $V_{D,M1}$ partagent donc le même noeud. Les deux opérateurs *OPVGD* et *OPVG* calculent le même paramètre et entrent donc en conflit. En introduisant un degré de liberté supplémentaire le graphe peut être transformé en un nouveau graphe, sans conflit.

6.2.3. Résolution du conflit

Pour résoudre le conflit qui est apparu entre les deux opérateurs définissant un paramètre, nous proposons de modifier le graphe par la technique de séparation des noeuds. On commence par choisir un opérateur pivot. Ici l'opérateur pivot est défini soit comme l'opérateur qui calcule une tension de grille(i.e. *OPVG*), soit comme celui qui calcule la tension de source (i.e. *OPVS*) d'un transistor MOS. en supposant que les tensions de décalage sont associées à une grille ou à une source de transistor MOS. Les transistors montés en diode sont donc exclus. Une fois le pivot choisi, les paramètres qui créent le conflit sont différenciés en introduisant un nouveau noeud. Les relations associées à chacun des opérateurs en conflit sont modifiées pour pointer vers les noeuds appropriés. On corrige éventuellement les relations de dépendance associées au noeud conflictuel

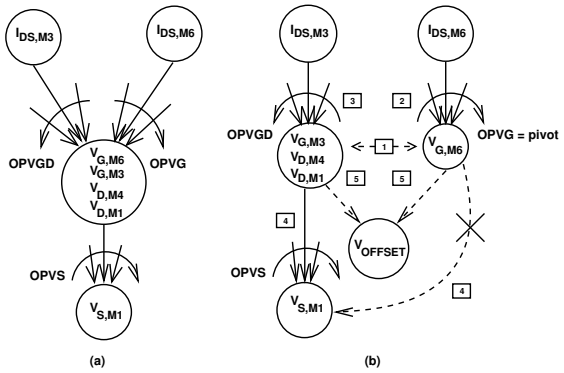

Figure 15: *Conflits entre opérateurs: (a) Détection, (b) Résolution. Les rectangles montrent les étapes de résolution.*

initial. On introduit un nouveau noeud, qui représente la tension de décalage. L'introduction de ce nouveau noeud permet de modifier le graphe de dépendance initial conflictuel en un graphe sans conflit. La connaissance exprimée par le graphe modifié est alors cohérente.

Dans la figure 15(b), l'opérateur pivot est *OPVG* car il calcule une tension de grille. Après la création d'un nouveau noeud (étape 1), le noeud initial n'est associé qu'à $V_{G,M6}$ et le nouveau noeud à $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. Comme *OPVG* avait été choisi pour calculer $V_{G,M6}$, les arcs de dépendance associés sont modifiés (étape 2) pour relier $V_{G,M6}$. *OPVGD* est modifié (étape 3) pour être relié à $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. Dans l'étape 4, l'arc entre $V_{S,M1}$ et le noeud initial est supprimé. Comme $V_{S,M1}$ dépend des paramètres de M_1, il faut modifier l'arc depuis $V_{G,M6}$ qui es indépéendnt des paramètres de M_1. Ainsi l'arc de dépendance entre $V_{S,M1}$ et le noeud initial $V_{G,M6}$ est remplacé par un arc de dépendance avec le nouveau noeud $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$ qui contient $V_{D,M1}$. Dans la cinquième étape on ajoute le noeud qui représente la tension de décalage. Il est relié au noeud initial et au noeud ajouté. Sa valeur est $V_{OFFSET} = V_{G,M6} - \{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. On obtient alors un graphe modifié sans conflit.

6.2.4. Graphe de dépendance du module

La figure 16 présente le graphe de dépendance de l'amplificateur à 2 étages. On suppose que $W_{M8} = W_{M5}$. Le transistor M_8 appartient au circuit de polarisation de l'amplificateur. Nous choisissons de spécifier le courant I_{BIAS}. Le courant de référence I_{REF} est un résultat du dimensionnement.

On souhaite satisfaire la contrainte $V_{G,M3} = V_{D,M4}$ en conservant le choix de $V_{eg,CM}$ et $V_{eg,M6}$. Dans la suite on va détailler l'analyse du graphe (Fig. 16) et expliquer la détection et la résolution du conflit :

1. Les paramètres d'entrée pour dimensionner l'amplificateur sont : $TEMP$, V_{DD}, V_{SS}, I_{BIAS}, $V_{eg,CM}$, L_{CM}, $V_{eg,DP}$, L_{DP}, V_{INCM}, V_{OUTCM}, $V_{eg,M5}$, $L_{\{M8,M5,M7\}}$, $V_{eg,M6}$, L_{M6} et le rapport

des courants entre les deux étages K. Ils apparaissent dans les noeuds entourés par un *rectangle*.

2. Les paramètres utilisés pour propager des valeurs sont représentés par les noeuds entourés d'un *cercle*. Considérons par exemple le paramètre (C2,veg_{cm},61) : il propage le paramètre de l'amplificateur (C1,$V_{eg,CM}$,67) au paramètre du miroir de courant (C3,$V_{eg,CM}$,8).

3. Les paramètres des dispositifs élémentaires sont propagés aux paramètres des transistors qui constituent le dispositif. Considérons par exemple le paramètre du miroir de courant : (C3,$V_{eg,CM}$,8) est propagé à M_3 via (C4,$V_{eg,M3}$,7) et à M_4 via (C4,$V_{eg,M4}$,7). Notons que M_3 et M_4 partagent la même tension effective de grille (C4,V_{eg},7).

4. Comme on a spécifié $V_{eg,M3} = V_{eg,CM}$, $V_{G/D,M3}$ est calculé par l'opérateur *OPVGD($V_{eg,M3}$)* en (C5,$V_{G/D,M3}$,2).

5. Comme on a spécifié $V_{eg,M6}$, $V_{G,M6}$ est calculé par l'opérateur *OPVG($V_{eg,M6}$)* en (C7,$V_{G,M6}$,1).

6. Les points (4) et (5) entrent en conflit avec l'hypothèse $V_{G,M6} = V_{D,M4}$. On applique donc la méthode de séparation des noeuds et on ajoute le noeud (C8,V_{OFFSET},0) pour calculer la tension de décalage, qui dépend du noeud initial (C7,$V_{G,M6}$,1) et du nouveau noeud (C5,$V_{G/D,M3}$,2). Lors du parcours du graphe, on évalue la tension de décalage comme la différence des tensions à ces deux noeuds.

7. Les largeurs des transistors sont calculées aux noeuds (C8,$\{M_1, M_2\}$,21), (C8,$\{M_3, M_4\}$,11), (C8,M_6,31), (C7,$\{M_5, M_8\}$,36) et (C8,M_7,25).

8. Comme $W_{M8} = W_{M5}$, ces paramètres partagent le même noeud : (C7,$\{M_5, M_8\}$,36).

9. Le courant de référence I_{REF} est calculé par l'opérateur *OPIDS(V_G,V_S)* au noeud (C8,$I_{DS,M8}$,41).

10. Le concepteur peut écrire des procédures de calcul qui font partie du graphe. Par exemple, noeud (C3,*l_dp*,56) est un paramètre de propagation calculé à partir de la procédure $IDS\_DP(I_{BIAS})$ écrite par le concepteur.

11. Le dimensionnement est effectué par la propagation des paramètres dans le graphe:

    (a) $V_{G,M3}$ et $V_{D,M3}$ sont calculés par *OPVGD($V_{eg,M3}$)* au noeud (C5,$V_{G/D,M3}$,2).

    (b) $V_{D,M3}$ est utilisé pour calculer $V_{D,M5}$, qui est égal à $V_{S,M1}$, avec l'opérateur *OPVS($V_{eg,M1}$,$V_{B,M1}$)* au noeud (C6,$V_{S,M1}$,24).

    (c) puis $V_{D,M5}$ est utilisé pour calculer $V_{G,M5}$ avec *OPVG($V_{eg,M5}$)* au noeud (C7,$V_{G,M5}$,29).

    (d) $V_{G,M5}$, qui est égal à $V_{G,M8}$, est utilisé pour calculer le courant I_{REF} avec l'opérateur *OPIDS($V_{G,M8}$,$V_{S,M8}$)* au noeud (C8,$I_{DS,M8}$,41).

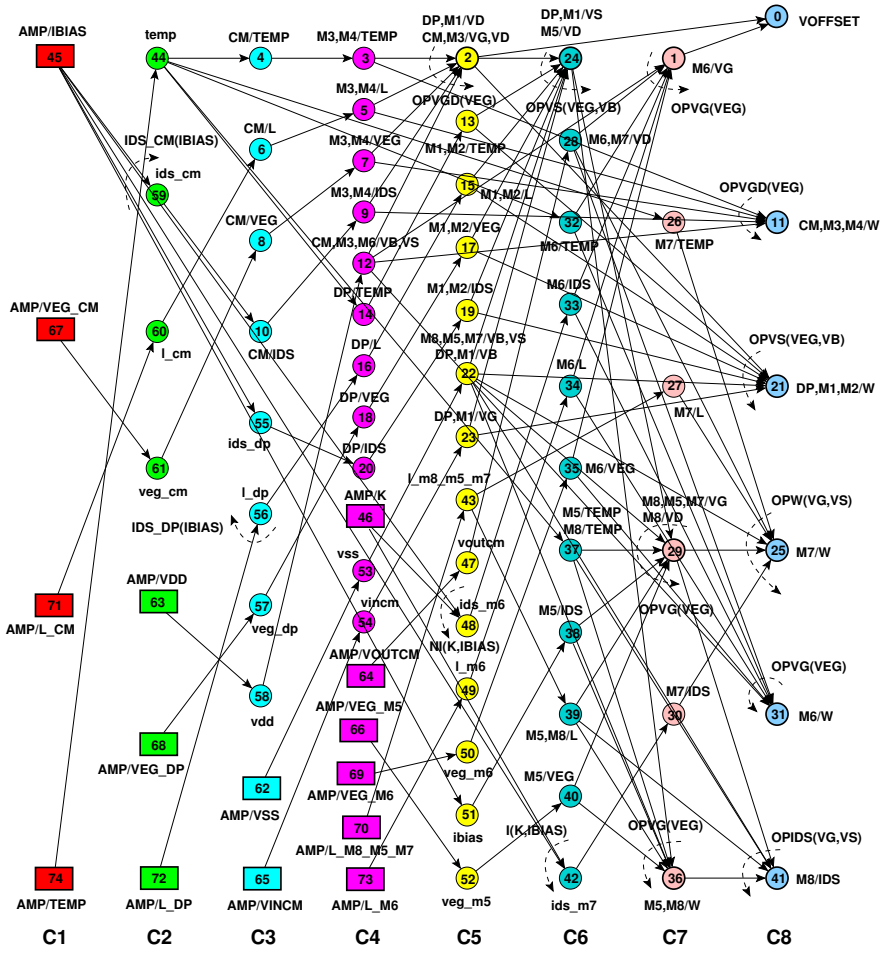

Figure 16: *Le graphe de dépendance du module amplificateur: (a) les rectangles contiennent les paramètres d'entrée, (b) les cercles fins sans arcs, les paramètres utilisés pour propager les valeurs, (c) les cercles en gras contiennent les opérateurs, (d) les cercles fins avec arcs, des procédures de calcul écrites par le concepteur. Chaque noeud est représenté par le triplet (colonne, nom, index) .*

6.2.5. Validation par simulation électrique

On a réalisé la simulation électrique de l'amplificateur, dimensionné suivant le graphe de la figure 16 et rebouclé par un gain unitaire. Le tableau 2 présente les résultats du point de fonctionnement, obtenus par dimensionnement puis par simulation. On peut calculer la tension de décalage ramenée à l'entrée en utilisant l'équation 6, qui donne la valeur -0.20424 mV avec les équations plus précises d'OCEANE [Porte08] on obtient la valeur -0.2047 mV. Cette valeur de la tension de décalage équilibre effectivement l'amplificateur. La tension de mode commun de sortie $V_{OUT} = V_{DD} + V_{DS,M6}$ vaut alors $0.6V$ pour une tension de mode commun d'entrée de $0.6V$. Remarquons que le point de fonctionnement obtenu par simulation électrique montre que M_1 et

M_2 diffèrent légèrement sous l'influence de la tension de décalage qui a été ramenée à l'entrée.

Table 2: *Polarisation et dimenionnement autaomatique de l'amplificateur Miller avec tension de décalage systématique en technologie CMOS 130NM et V_{DD} = 1.2V.*

Paramètre	Synthèse		Simulation		
	M_1,M_2	M_3,M_4	M_1	M_2	M_4
$I_{DS}(\mu A)$	50.0	-50.0	50.028	49.971	-49.971
$V_{GS}(V)$	0.453075	-0.462552	0.45317	0.45297	-0.4626
$V_{DS}(V)$	0.590524	-0.462552	0.59057	0.6099	-0.44328
$V_{BS}(V)$	-0.146925	0.0	-0.14683	-0.14683	0.0
$V_{th}(V)$	0.333076	-0.342552	0.33304	0.33304	-0.34255
$V_{eg}(V)$	0.12	-0.12	0.12013	0.11993	-0.12005
$V_{dsat}(V)$	0.115618	-0.120473	0.1157	0.11557	-0.12051
$g_m(mA/V)$	0.671032	0.653846	0.67095	0.67075	0.65333
$g_{ds}(\mu A/V)$	4.24831	2.85842	4.2497	4.1989	2.9923
$g_{mb}(mA/V)$	0.13131	0.143927	0.13129	0.13127	0.14382
$C_{gd}(fF)$	10.8436	30.2681	10.830	10.818	30.443
$C_{gs}(pF)$	0.165409	0.523254	0.16524	0.16518	0.52325
$C_{sd}(fF)$	0.0378348	0.343275	0.037829	0.034159	0.39329
$C_{bd}(fF)$	0.0341236	0.284976	0.034119	0.030809	0.3265
	M_6		M_6		
$I_{DS}(\mu A)$	-500		-500.07		
$V_{GS}(V)$	-0.443267		-0.44328		
$V_{DS}(V)$	-0.6		-0.6		
$V_{i,off}(mV)$	-0.20424[1]		–		
$V_{i,off}(mV)$	-0.204734[2]		-0.204734		

1. Avec Eq. (6).

2. Avec les équations d'OCEANE [Porte08].

7. Synthèse et connaissance du concepteur

Ce paragraphe résume le chapitre 8 de la thèse qui montre comment exploiter la méthode de dimensionnement par calcul du point de polarisation pour résoudre le problème d'optimisation présenté à la figure (5).

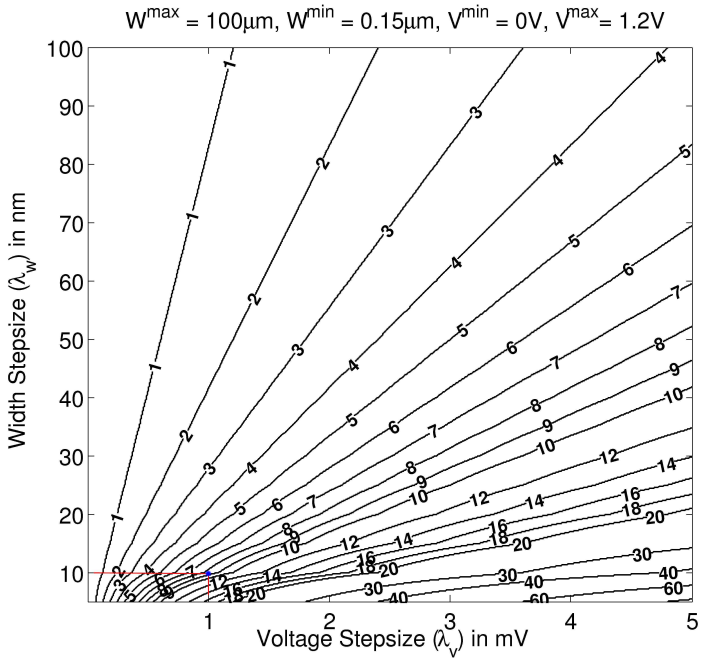

Figure 17: *Facteur de réduction par variable.*

7.1. Choix des variables de synthèse

Dans l'idée d'évaluer la réduction de l'espace de conception introduite par le changement de variables à optimiser, nous définissons une figure de mérite appelée *facteur de réduction* qui définit le rapport entre le nombre de valeurs possibles pour les largeurs $W = W_i^{min} : W_i^{max} : \lambda_i^w$ et celui des tensions $V = V_i^{min} : V_i^{max} : \lambda_i^v$,

$$FacteurDeRéduction = \prod_{i=1}^{n} \frac{\lambda_i^v}{\lambda_i^w} \cdot \frac{W_i^{max} - W_i^{min}}{V_i^{max} - V_i^{min}} \tag{7}$$

où λ_i^w est l'incrément de la largeur qui varie entre W_i^{min} and W_i^{max} et λ_i^v et celui de la tension qui varie V_i^{min} and V_i^{max}. Nous voyons que le facteur de réduction est souvent bien supérieur à un. En effet, l'évolution des technologies fait décroître les tensions d'alimentation $V_i^{max} - V_i^{min}$ et le pas de grille fondeur λ_i^w. Ce facteur de réduction est représenté sur les courbes de la figure pour un cas mono-dimensionnel, i.e. $n = 1$. En utilisant l'équation (7) pour $n > 1$, nous pouvons obtenir un facteur de réduction important, et donc une réduction importante de l'espace de conception en remplaçant la variable *largeur* par la variable *tension de polarisation*.

Pour tirer partie de la réduction potentielle de l'espace de conception, nous avons défini une méthode de synthèse en trois phases, qui s'appuie sur la décomposition hiérarchique en dispositifs

élémentaires et modules. La première phase utilise des algorithmes classiques comme *Nelder-Mead Simplex*[Nelder65, Lagarias98] pour donner des valeurs aux composantes du premier vecteur de la figure (5). La deuxième phase calcule les deuxième et troisième vecteurs à l'aide du plan de dimensionnement. Enfin, les performances sont évaluées par des équations, disponibles sous OCEANE [Porte08].

7.2. Application

Cette méthode a été appliquée à l'amplificateur Miller (deux étages non-différentiel) de la figure (14). Le plan de dimensionnement est toujours représenté par le graphe de dépendance donné par la figure (16).

Le tableau 3 donne les spécifications de l'amplificateur, ainsi que les résultats de synthèse et de simulation dans une technologie CMOS 130nm . Ces résultats montrent que cette approche donne très rapidement un résultat comparable, en termes de précision, à la simulation. Le temps d'exécution moyen est de 76 secondes sur un Intel Centrino 1.7GHz avec 2MB de mémoire cache. Ces résultats sont satisfaisants si on les compare avec d'autres études : 16 minutes pour ASTRX/OBLX[Ochotta96], 76 minutes pour ASF[Krasnicki01], 2.8 heures pour ANACONDA[Phelps00].

7.3. Réutilisation : changement de spécifications

Le circuit paramétrable ainsi conçu constitue un bloc IP AMS qu'il est facile de réutiliser pour une autre application, ayant d'autres spécifications. Supposons maintenant que l'on ait besoin d'un amplificateur de fréquence unitaire double du cas précédent. On élargit alors l'intervalle admissible du courant I_{BIAS}. Le tableau 4 présente la comparaison de résultats issus de la synthèse et des résultats de simulation. La fréquence de transition doit être supérieure à 12Mhz et I_{BIAS} peut atteindre 60μA. Les résultats montrent effectivement que, par rapport au cas précédent, I_{BIAS} a augmenté pour satisfaire la contrainte sur la fréquence de transition (tableau 4). Le temps de calcul moyen sur Intel Centrino 1.7GHz avec 2MB est 87 secs. Ce qui illustre l'efficacité de cette méthode.

8. Conclusions

Cette thèse a présenté une méthode pour dimensionner un circuit en calculant son point de polarisation. Cette approche permet de déduire automatiquement une procédure de dimensionnement quelle que soit la topologie électrique cible. La procédure de dimensionnement s'exprime par un graphe de dépendance permettant de propager des contraintes de façon hiérarchique. Cette méthode a été appliquée avec succès pour dimensionner plusieurs types de circuits analogiques.

Les algorithmes développés dans cette thèse ont été intégrés à l'environnement de conception CAIRO+ du LIP6.

Cette approche, combinant optimisation et connaissance, permet d'accélérer la synthèse de circuits analogiques, tout en garantissant une précision comparable à celle d'un simulateur électrique. Elle apporte une contribution à la synthèse de circuits analogiques fondée sur la connaissance du concepteur.

Table 3: *Dimensionnement et optimisation en technologie 130nm CMOS avec V_{DD} = 1.2V. Résultats et validation par simulation.*

Circuit Performances	Spécifications	Synthèse[1]	Simulation
Gain AC (dB)	> 65	71.22	70.83
Gain de mode commun (dB)	< 17	5.59	8.82
Fréquence de transition (MHz)	> 6	10.79	10.66
Marge de phase (degrés)	> 76	76.69	76.9
Tension de décalage à l'entrée (mV)	< 2	-1.03	-1.07
Bruit à l'entrée @1Hz ($\mu V/\sqrt{Hz}$)	< 20	10.86	10.14
Bruit à l'entrée @UGF ($\mu V/\sqrt{Hz}$)	< 0.02	0.014	0.015
Slew Rate ($V/\mu s$)	> 6	8.25	9.16
Transistors en Saturation	$= 8$	8	8
Puissance (mW)	< 1	0.176	0.176
Réjection de mode commun (dB)	–	65.63	62.01
Tension minimale à l'entrée (V)	–	0.53	0.53
Tension maximale à l'entrée (V)	–	1.04	1.04
Tension maximale à la sortie (V)	–	1.12	1.12
Tension minimale à la sortie (V)	–	0.10	0.10
Surface (μm^2)	–	69	69
Temps de calcul moyen sur 10 runs (secs)	-	76	-
Paramètre à optimiser	Domaine de variation	Synthèse	
$L_{M_5} = L_{M_7} = L_{M_8}(\mu m)$	0.2:3:0.1	0.6	
$L_{CM}(\mu m)$	0.2:3:0.1	0.6	
$L_{M6}(\mu m)$	0.2:3:0.1	0.2	
$L_{DP}(\mu m)$	0.2:3:0.1	0.8	
V_{eg,M_5} (V)	0.01:0.2:0.01	0.1	
$V_{eg,DP}$ (V)	0.01:0.2:0.01	0.08	
$V_{eg,CM}$ (V)	-0.2:-0.01:0.01	-0.15	
V_{eg,M_6} (V)	-0.2:-0.01:0.01	-0.04	
$I_{BIAS}(\mu A)$	10:30:1	25.0	
K	1:5:0.5	4.5	
C_C (pF)	1:5:0.1	2.9	
Paramètre constant	Valeur	Synthèse	
$TEMP$ (degrés)	300.15	300.15	
V_{DD} (V)	1.2	1.2	
V_{SS} (V)	0.0	0.0	
V_{ICM} (V)	0.6	0.6	
V_{OUTCM} (V)	0.6	0.6	
C_{LOAD} (pF)	3.0	3.0	

1. Synthèse effectuée avec génération de dessins des masques

Table 4: *Dimensionnement et optimisation en technologie 130nm CMOS with V_{DD} = 1.2V changement de spécifications*.

Performances	Spécifications	Synthèse[1]	Simulation
Gain AC (dB)	> 65	72.04	71.49
Gain de mode commun (dB)	< 17	15.91	18.13
Fréquence d transition (MHz)	$> \underline{12}$	17.2	17.09
Marge de phase (degrés)	> 76	76.44	76.59
Tension de décalage à l'entrée (mV)	< 2	-0.67	-0.6882
Bruit à l'entrée @1Hz (μV/$\sqrt{Hz}$)	< 20	8.31	7.02
Bruit à l'entrée @UGF (μV/$\sqrt{Hz}$)	< 0.02	0.0125	0.0127
Slew Rate (V/μs)	> 6	18.61	20.0
Transistors en Saturation	$= 8$	8	8
Puissance (mW)	< 1	0.432	0.432
réjection de mode commun (dB)	–	56.13	53.36
Tension min mode commun entrée (V)	–	0.58	0.58
Tension max mode commune entrée (V)	–	1.03	1.03
Tension sortie max (V)	–	1.1	1.1
Tension sortie min (V)	–	0.09	0.09
Surface (μm^2)	–	155	155
Temps de calcul moyen sur 10 runs (secs)	-	87	-
Paramètre à optimiser	Domaine de variation	Synthèse Synthèse	
$L_{M_5} = L_{M_7} = L_{M_8}(\mu m)$	0.2:3:0.1	0.3	
$L_{CM}(\mu m)$	0.2:3:0.1	0.9	
$L_{M6}(\mu m)$	0.2:3:0.1	0.3	
$L_{DP}(\mu m)$	0.2:3:0.1	1.8	
V_{eg,M_5} (V)	0.01:0.2:0.01	0.07	
$V_{eg,DP}$ (V)	0.01:0.2:0.01	0.17	
$V_{eg,CM}$ (V)	-0.2:-0.01:0.01	-0.17	
V_{eg,M_6} (V)	-0.2:-0.01:0.01	-0.09	
$I_{BIAS}(\mu A)$	10:$\underline{60}$:1	56	
K	1:5:0.5	5	
C_C (pF)	1:5:0.1	2.6	
Paramètre constant	Valeur	Synthèse	
$TEMP$ (degrés)	300.15	300.15	
V_{DD} (V)	1.2	1.2	
V_{SS} (V)	0.0	0.0	
V_{ICM} (V)	0.6	0.6	
V_{OUTCM} (V)	0.6	0.6	
C_{LOAD} (pF)	3.0	3.0	

1. Synthèse effectuée avec génération de dessins des masques

Table 5: *Comparison of Synthesis Tools [Ochotta98].*

	Synthesis Tool							
	OPASYN [Koh87]	**OASYS [Harjani87]**	**IDAC [DeGrauwe84b]**	**ARIADNE [Swings91a]**	**STAIC [Harvey92]**	**ISAID [Makris92]**	**Maulik [Maulik91]**	**AMGIE [Plas01]**
School	University of California at Berkeley	Carnegie Mellon University	Centre Suisse d'Electronique et de Microelectronique	Katholieke Universiteit at Leuven	University of Waterloo	Imperial College in London	Carnegie Mellon University	Katholieke Universiteit at Leuven
Performance evaluation	Equations	Equations	Equations	Equations	Equations	Equations/ Qualitative Reasoning	Equations	Equations
Non-Linear device models	Simplified equations	Simplified equations	Custom models	Simplified equations	Simplified equations	Simplified equations	Equations+BSIM	Equations+BSIM
Worst-case accuracy	200%	25%	15%	10%(low-perf.)	24%	14%	24%	Not reported
Search methods	Gridded, Steepest descent optim.	Plan steps with backtracking	Plan steps with post optim.	Simulated annealing	Coarse initial optim. + detailed final optim.	Qualitative reasoning + post optim.	Sequential quadratic programming (SQP)	VFSR, Hooke-Jeeves, minimax, or SQP
Synthesis time (approximate)	1 min	5 sec	A Few seconds	5 min	3 min	Not reported	1 min	few minutes to 2 hour
Machine	VAX 8800	VAX 8800	CPU N/A	CPU N/A	MIPS 2000	Not reported	DEC 3100	SUN Ultra 1-170, HP 712/80 UNIX
Preparatory effort to add new circuit	2 weeks for well understood circuit	6 months including circuit analysis	4-45 designer-months	Not reported	3 circuits required 100000 lines of code	Not reported	6 months including circuit analysis	1 week
Equations derived	Manually	Manually	Symbolic simulation + Manually	Symbolic simulation + Manually	Manually	Manually	Manually	Manually + Symbolic analysis
How/Where Equations are stored	Hard-coded	Hard-coded	Hard-coded + Design files	Not reported	language + Database	Not reported	Hard-coded	Hard-coded
Most Complex circuit example	7 variables (60 device opamp)	19 variables (17 device opamp)	N/A (15 circuit types, incl. delta-sigma, from 5-30 devices)	14 variables (9 device opamp)	N/A (22 device opamp)	8 variables (13 device opamp)	39 variables, 7 for topology selection (19 devices opamp)	N/A (Particle Detector Front-End)

Chapter 1

Introduction

1.1 Motivation

The complexity of integrated electronic circuits being designed is continuously increasing as advances in process technology make it possible to create mixed-signal integrated SoC designs. Most parts of these SoC's are completely digital rather than analog. Today, the transistor density has reached 820 million transistors for latest Intel quad-core processors, fabricated in 45nm technology. This huge increase in complexity has been made possible due to the development of synthesis, layout and verification tools for the digital domain. The identification of intermediate design representations for describing digital subsystems were behind the success of these tools: A subsystem is described using VHDL or Verilog behavioral description. Then this behavioral description is converted into an intermediate design representation, namely *control/data flow graphs*. This intermediate representation allows the abstraction of behavior into a clocked sequence of operators. The control/data flow graphs are then optimized to more compact graphs. The sequencing of operators in the graph is implemented using a finite state machine called *control path*. In each state, the flow of data and operations are then implemented by connecting standard cells with a bus architecture. The resulting hardware is called *the data path*. Both the control path and data path realize together the required behavior. It is clear that the successful identification of intermediate design representations gave the EDA tools enough insight to deal with the problems of digital design and behavioral abstraction.

Despite some substantial progress achieved in the past from academic research, the specific problems of analog circuit synthesis are such cumbersome that some designers strongly doubt that it will ever happen. The fundamental problem for analog design is the lack of an intermediate design representation that can abstract the behavior of analog circuits in appropriate sequence of design steps. Spice netlists are the most common design representation for analog circuits. Netlists describe structure rather than behavior. In general, a design representations for analog circuits should have the following characteristics:

- The design representation should allow design abstraction from the system level down to

the technology level.

- It should abstract the behavior of the circuit, hierarchically, from the atomic behavior of building blocks.
- It should allow to compute the effect of system-level parameters on the transistor-level parameters.
- It should allow the documentation of design knowledge in appropriate design steps.
- It should ensure that the documented knowledge is an executable, reusable and consistent entity.
- It should point the different problems in analog design and repair strategies to resolve them.
- It should allow the designer to use more intuitive quantities to describe the behavior.
- It should allow the computation of the electrical behavior of the analog circuit accurately and efficiently.
- It should be used for many different types of analyses such as DC sweep, Monte-Carlo analysis, sensitivity analysis, ... etc.
- It should abstract the behavior independent of the fabrication process used.
- It should be directly mapped to any fabrication process.
- It should be stored in a database for later use.

Despite that few successful attempts have been recorded in literature, the hope is to be able to write soft analog intellectual properties (IP) that can be synthesized from the system level specifications down to the physical level given a target technology. The emergence of analog intellectual properties (IP) will definitely favorize the development of this research direction. Design reuse of analog IP blocks will thus gain more importance in the coming few years especially with the rapid advances in fabrication technologies led by the digital system needs. Analog cells would have to be migrated to these new technologies with minimal manual contributions. While analog design automation methodologies are not yet widely accepted by analog designers, design reuse will soon be a huge driving force.

1.2 Contribution

The main contribution of this work is the identification of an intermediate representation for documenting analog design knowledge into a consistent and reusable form. The proposed design representation is quite general since it can describe any type of analog circuits. It is evolutive

since it promotes the enhancement, the modification and the maintenance of analog intellectual properties. It possesses all the advantages enumerated in the previous section. To summarize, the work presented in this thesis is fivefold:

Transistor Modeling: In this phase, a set of electrical parameters have been chosen to size and bias a transistor. Since it is required to compute these parameters, it is essential to numerically invert a BSIM3V3 transistor model in order to compute them accurately. This step is necessary to control design errors in the earliest design phases. These design errors may accumulate to other sources of errors in later design phases, resulting in an erroneous design. This phase has been successfully implemented giving rise to the concept of the *sizing and biasing operators*.

Design Methodology: An innovative design approach to extract and reuse design knowledge is proposed. It is called *hierarchical sizing and biasing* methodology. Traditional analog design requires a complex sizing procedure, called a *design plan*, in order to compute transistor biases and geometries from circuit specifications. One circuit has many possible design plans depending essentially on designer expertise and on hypotheses used in the design. Furthermore, the abstract design may slightly shift from a simulated circuit design. We propose the hierarchical sizing and biasing to automatically extract abstract and simulatable design plans. The abstract design plan is generated in the *designer mode*. This design plan sizes and biases the analog circuit while respecting constraints and hypotheses imposed by the designer. The simulatable design plan is generated in the *simulator mode*. It allows the designer to verify if the sized circuit resulting from his abstract design plan is functioning as expected. Both abstract and simulatable design plans are extracted automatically from the circuit structure and designer hypotheses. The method ensures that the generated design plans represent reusable and consistent entity. Our approach deals with different aspects of analog design such as identification of degrees of freedom, systematic offsets and negative feedback circuits. The proposed method proved its efficiency and accuracy over a wide range of circuit designs.

Language Development: The implementation of the hierarchical sizing and biasing methodology have been studied thoroughly. This has led to the development of a design space exploration language, which has been integrated as part of the CAIRO+ framework. The language has the following characteristics:

- It is based on text files, hence, it allows modification, debugging and maintenance of analog intellectual properties.
- It allows the designer to document his analog design knowledge intuitively and seamlessly.
- It allows the designer to express design constraints and hypotheses very easily.

- It automatically extracts design knowledge while respecting designer's constraints and hypotheses.
- It allows the designer to ensure knowledge consistency.
- It suggests to the designer possible repair strategies to deal with inconsistency.
- It synthesizes the analog IP using a hierarchical bottom-up approach.
- It controls the execution of knowledge using a top-down (left-to-right) approach.
- It accurately reports design errors that occurs during design analysis and execution.
- It ensures the independence of the design representation from the fabrication technology.
- It couples the design representation with a layout generation phase to ensure that the generated layout satisfies designer's constraints and hypotheses.
- It creates a customized graphical influence exploration tool for the analog IP.
- It controls the execution of optimization algorithms used to optimize the analog IP.
- It configures testbenchs used to instantiate and test the analog IP.

Tool architecture: A new tool architecture has been proposed to demonstrate the strength of the hierarchical sizing and biasing methodology. The architecture implements a language-based methodology that targets both automated synthesis and design reuse for analog IPs. The architecture alleviates the effort of knowledge documentation by introducing a minimum level of design automation that is still acceptable and fully controlled by the designer.

Case Studies: The hierarchical sizing and biasing methodology was successfully applied to five different analog intellectual properties IP, namely:

- A two-stage single-ended operational transconductance amplifier.
- A fully differential current-mode integrator.
- A fully differential common-mode feedback amplifier.
- A fully differential transconductor.
- A 0.5V fully differential bulk-input amplifier with local common-mode.

Many aspects of analog design are considered throughout the examples. In each intellectual property, a complete design plan is automatically generated and represented using dependency graphs as an intermediate representation for analog design knowledge. The consistency of the

design plan is ensured. Otherwise, repair strategies are proposed to transform the design into a reusable and consistent entity. During the synthesis phase, the design plan is executed to determine transistor sizes and biases. Those are later used for analog simulation to ensure their correctness and accuracy. Both synthesis and simulation have demonstrated high agreements, proclaiming that the proposed methodology is capable of producing simulator quality designs in a very reasonable amount of execution time. Thus, facilitating interactive analog design. The presented analog intellectual properties have been synthesized and simulated in $0.13\mu m$ technology process with $V_{DD} = 1.2V$.

1.3 Outline

This section gives a brief overview of the contents of the following chapters:

After a brief introduction in chapter 1, chapter 2 defines the context of the thesis. The problem definition and the objectives of the thesis are clearly stated.

Chapter 3 introduces the state-of-art of the different research fields related to the thesis work. It includes the methods of DC operating point computation, compact device modeling, model development and standardization efforts, design reuse techniques for analog IP and analog design representation.

In chapter 4, a complete formulation for the transistor sizing and biasing that unifies both standard simulation method and operating point driven formulation, will be presented. In the proposed formulation, a library of procedures for computing the sizes and biases of a transistor is developed. These procedures numerically invert the standard BSIM3V3 transistor model. The procedures are overloaded to implement both standard simulation method and operating point driven formulation. A sizing and biasing example is demonstrated at the end of the chapter.

Chapter 5 elaborates a design representation for the basic building blocks called *devices*. Some key concepts are elaborated and used to create suitable sizing procedures for the devices. These sizing procedures are represented by *dependency graphs* which are chosen to be the intermediate design representation for devices. Examples are illustrated throughout this chapter.

In Chapter 6, the hierarchical sizing and biasing methodology is proposed to automatically generate suitable *design plan* for the circuit topology. The created design plan respects the designer's hypotheses. The design plan is represented by *dependency graphs*. Since a design plan should represent a consistent knowledge about the circuit, it should not contain any inconsistency. Inconsistency occurs if the design is *under-* or *over-specified*. Inconsistency appears as *redundant*, *cyclic* or *conflicting* hypotheses. The principles presented in this chapter deals with each type of inconsistency. Based on those principles, different aspects of analog designs are then studied such as degrees of freedom, systematic offset, feedback circuits, ... etc . Then, some automated approaches are proposed to deal with these aspects. As a case study, the proposed methodology is illustrated through the design of an OTA amplifier.

In Chapter 7, the proposed methodology is applied on four more case studies, namely: a fully differential current-mode integrator, a fully differential common-mode feedback amplifier, a fully differential transconductor and a 0.5V fully differential bulk-input amplifier with local common-mode. For each circuit, the design plan is automatically generated and the circuit is synthesized. The computed sizing and biasing are then compared to the simulation results of a commercial analog simulator.

In chapter 8, the concept of knowledge-aware synthesis is introduced as a potential application for the hierarchical sizing and biasing methodology. The advantages of knowledge-aware synthesis are discussed. It will be shown that the method allows more intuitive choice of optimization parameters, potential reduction in the design space, and faster optimization performance when introduced inside an optimization loop. A complete formulation of the cost function along with the *Nelder-Mead Simplex* optimization algorithm will be discussed. It will be shown that the proposed synthesis system is capable of producing simulator-like quality designs in a very reasonable amount of execution time. This compares favorably to the state-of-art synthesis tools. Consequently, interactive analog design can be considered using the proposed synthesis system.

In chapter 9, conclusions are drawn together with the possible directions for future research and development.

Chapter 2

Motivation and Problem Definition

2.1 Introduction

Analog design automation tools lack behind digital tools in generality and productivity. The fundamental difficulty is how to represent analog design knowledge so that analog design automation tools have enough insights on design issues. This chapter presents the motivations that led to the development of the hierarchical sizing and biasing methodology.

In section 2.2, the motivations driving our study are introduced. It is shown that firm intellectual properties (IP) are considered the most appropriate format for analog IP cores.

In section 2.3, an audio digital signal processor (DSP) will be discussed as a design example. The low-pass filter that is one important component of the audio processor is realized. Traditional design phases will be outlined out of the low-pass filter design.

In section 2.4, motivations and requirements on the hierarchical sizing and biasing methodology will be outlined.

In section 2.5, a tool architecture is proposed to satisfy all the requirements for the hierarchical sizing and biasing methodology.

In section 2.6, conclusions about analog design automation and firm intellectual properties are drawn.

The hierarchical sizing and biasing methodology will be explained in further details in the next chapters.

2.2 Motivation: System-on-Chip Reuse and Integration

The semiconductor industry has continued to make impressive improvements in the achievable density of very large-scale integrated (VLSI) circuits [Semiconductors]. In order to keep pace with the levels of integration available, design engineers have developed new methodologies and techniques to manage the increased complexity inherent in these large chips. One such emerging methodology is system-on-chip (SoC) design [Saleh06], wherein predesigned and preverified

blocks (often called intellectual property (IP) blocks, IP cores, or virtual components) are obtained from internal sources, or third parties and combined on a single chip. These reusable IP cores [Keating02] may include embedded processors, memory blocks, interface blocks, analog blocks, and components that handle application specific processing functions.

The main prerequisite for creating SoC designs is a set of reusable IP blocks that support plug-and-play integration. IP blocks are the highest level building blocks of a SoC. A library of reusable IP blocks with various timing, area, power configurations is the key to SoC success as the SoC integrator can apply the trade-offs that best suit the needs of the target application. The process of creating a reusable IP block, however, differs from the traditional ASIC design approach. Typically, it may require five times as much work to generate a reusable IP block compared to a single-use block [Keating02].

While design productivity can be improved significantly with the use of digital IP blocks, another bottleneck exists if the designs include analog and mixed-signal components. Digital design has a well-defined, top-down design methodology but analog/mixed-signal (AMS) design has traditionally been an ad hoc custom design process. When analog and digital blocks coexist on the same substrate, the analog portion of the design can be more time-consuming to develop even though it may represent a smaller percentage of the chip area. In a few years, it is anticipated that more than 70 percent of all SoC designs will have some form of analog/mixed-signal blocks [Semiconductors]. This increase is consistent with the expected growth of the wireless industry over the same period.

In order to keep pace with rapidly evolving markets, the productivity of AMS design can be improved using a mixed-signal SoC design flow [Kundert00, Gielen00], employing AMS IP [Madrid01, Li03, Hamour03]. One of the main advantages of the use of AMS IP in SoCs is the potential reduction in power, which is especially important in battery-operated applications such as personal digital assistants (PDAs), wireless local area networks (LANs), etc. Typical AMS components include operational amplifiers, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), phase- locked loops (PLLs), delay-locked loops (DLLs), serializer/ deserializer transceivers, filters, voltage references, radio frequency (RF) modules, voltage regulators, analog comparators, etc. Many of these blocks are delivered in the form of hard IP and targeted to one application in a specific fabrication technology. Therefore, they cannot be easily migrated to other applications and other technologies by the end user.

Compared to digital IP, AMS IP must provide an even greater degree of flexibility in the design parameters and performance characteristics. While the general function of an analog block may be the same in different applications, the design specifications may vary widely between the applications. Furthermore, the performance of AMS IP blocks is significantly influenced by parasitics and interactions with the surrounding environment, often in the form of power supply fluctuations and substrate noise effects. Proper modeling of the interfaces between the different blocks is important in the design process to account for different effects such as loading and coupling. This

is needed to achieve correct performance when used in the overall system-level design [Ju91].

Currently, because of the complexity of analog/mixed signal design and its sensitivity to the surrounding environment, AMS blocks are most commonly presented in the form of hard IP. However, this form has limited scope of applications. Hard IP will reduce the design cycle significantly when the specifications and fabrication processes are identical, but will not greatly improve the design cycle if it has to be modified in any way or migrated to a new process. This calls for a more flexible definition for the format in which the AMS IPs are provided. Firm IP [Saleh06] appears to be the most appropriate format to deliver the AMS IP library components. In this form, the IP captures suitable schematics of the analog blocks with parameters that are adjustable to optimize the design for specific applications. Unlike hard IP, this form allows ease of migration of IP from foundry to foundry, customer to customer, and application to application.

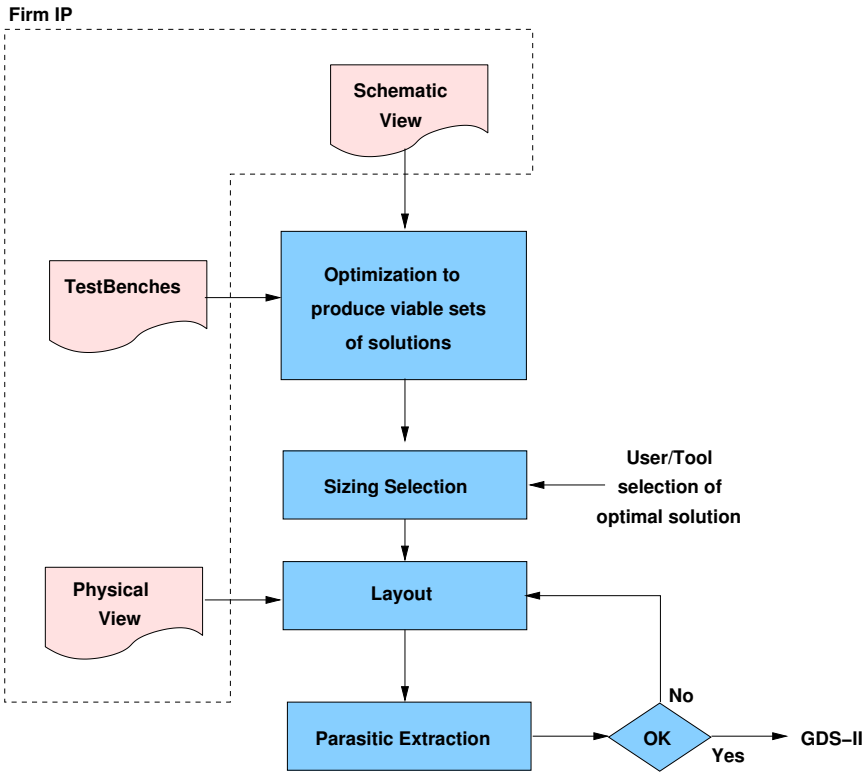

Figure 2.1: *Proposed analog/mixed-signal IP hardening flow.*

The traditional flow for AMS design relies heavily on the expertise and experience of the designer. The design process begins with the performance specification of the component for a target application. Ideally, the AMS block should be described at a high level in the form of soft IP as in the digital case. System-level designers typically use tools such as MATLAB [MathWorks] for specification and simulation. In addition, analog/mixed-signal hardware description languages (AMS-HDL) are increasingly used to model these types of circuits. The languages are a relatively

new addition to the design process, and the most commonly used are Verilog-AMS [Verilog-AMS] and VHDL-AMS [VHDL-AMS]. SystemC-AMS [SystemC-AMS] that covers functional, architectural and linear electrical network levels, is currently being standardized. Automatic generation of AMS architectures from AMS-HDL is still in its infancy because of the large number of variables associated with AMS design. However, the current contribution of these AMS-HDLs to system-level design is highly significant. They provide the necessary platform for system level verification, an important part of design quality. Verification of mixed-signal SoCs requires co-simulation of analog and digital behavioral models to reduce simulation costs [Rashinkar01].

Since AMS blocks cannot be easily synthesized from a high-level specification without low-level support, designers must follow a design process such as the firm IP hardening flow [Rajsuman00, Hamour03] illustrated in Fig. 2.1. The starting point of the flow is the set of selected library components that comprise the unoptimized schematic view of the design. This library consists of parameterized reusable components and is an essential part of the design flow. After an architecture is chosen, the firm IP is taken through the IP hardening flow for optimization of the circuit parameters to maximize performance and to generate the final GDSII layout of the block.

When developing firm reusable AMS cores [Madrid01, Hamour03], the usual precepts of a good design must be followed. That is, there should be a good formal specification, a good architectural design and a good circuit implementation. In addition, there are a number of steps that must be followed to achieve reusability, as discussed below. A successful IP block should be parameterized, easily verified through reusable test benches, well documented, and have associated views to ease the derivative design process. Specifically, an AMS IP block should have a behavioral/ analytical view (in AMS-HDL), a parameterized schematic view (transistor level), and a layout view (floor plan). In addition, test benches are needed to validate the performance of the circuit under different operating conditions and at various process corners. They are used as the basis for verification of specifications and for exploration of the design space for the system.

We conclude that the development of AMS IP must take a different approach compared to digital IP development. The IPs must be able to handle and transfer both design experience and heuristics from the original design to subsequent design derivatives. In reality, this constitutes the reusable IP in the analog design process.

In order to address the problem of layout representation for firm IP, a startup company named CIRANOVA [Ciranova] developed a language-based design methodology for analog and mixed-signal designers, enabling the creation of reusable and migratable layout generators. It is evident that CIRANOVA's approach requires a language-based behavioral description methodology that allows designers to develop analog IPs in the form of libraries of parameterized generators, to size and bias those IP, and to generate physical views for sized IPs. Both the electrical behavior and physical view can communicate in a loop in order to ensure that the generated physical view have the required electrical behavior. In this perspective, CAIRO+ [Dessouky01, Tuong06] language-

based framework was proposed to facilitate the development of parameterized generators for analog firm IP.

The work presented here, addresses the problem of elaborating an intermediate design representation in order to document, judge and repair knowledge in an analog firm IP. This intermediate representation will serve for sizing and biasing a firm IP automatically. The electrical view of a firm IP communicates with its physical view to accurately generate the corresponding hard IP. The design representation will provide future EDA tools enough insight to deal with different aspects of analog design. As anticipated, the design representation will deal with both design experience and heuristics imposed by the designer. This is performed in a fully abstract way suitable for analog firm IPs. Briefly, the designer will have full control on the construction of electrical and physical views of the analog firm IP.

2.3 Analog Design of an Audio DSP

In this section, we illustrate the traditional steps involved in the design of an audio DSP processor. We develop a top-down systematic approach for the design of a part of the audio DSP, i.e. a low-pass filter. To realize the filter, we use a single-stage amplifier in an active-RC configuration. We show how specifications in the system level affect circuit performances imposed on the amplifier. We deduce how circuit parameters are mapped in the design space. In later sections, we propose a hierarchical sizing and biasing methodology that reposes on those principles.

2.3.1 Case Study: Audio DSP

Suppose that the audio digital signal processor shown in Fig. 2.2 is to be designed. The audio processor should eliminate echos that accompany the singer sound. Instead of directly hearing the singer, the singer sound will be captured by a microphone and then processed by the audio processor. Then the processed sound is directed to loud speakers free of echos.

Let us further examine the architecture of the audio DSP. The singer sound is first captured using a microphone. The microphone converts the sound into an electrical signal. This signal is then filtered using low-pass filter to reject noise that is outside the audio band. The filtered signal is then fed to amplifier to amplify the signal level. This amplified signal is then converted to a bit stream using an analog-to-digital converter (ADC). Next, the bit stream is processed using a DSP processor that executes an echo elimination algorithm and outputs the sound in the form of a bit stream that is free of echo. The output bit stream is converted back to an analog signal using a digital-to-analog converter. This processed analog signal is filtered using a low-pass filter to keep the signal in the audio band and remove noise outside the band. The audio signal is then amplified using a power amplifier which drives loud speakers.

The effect of each block, beyond the DSP processor, on the audio signal is illustrated in Fig. 2.3. The output of the DSP processor is a repeated audio signal spectrum with sampling frequency F_s

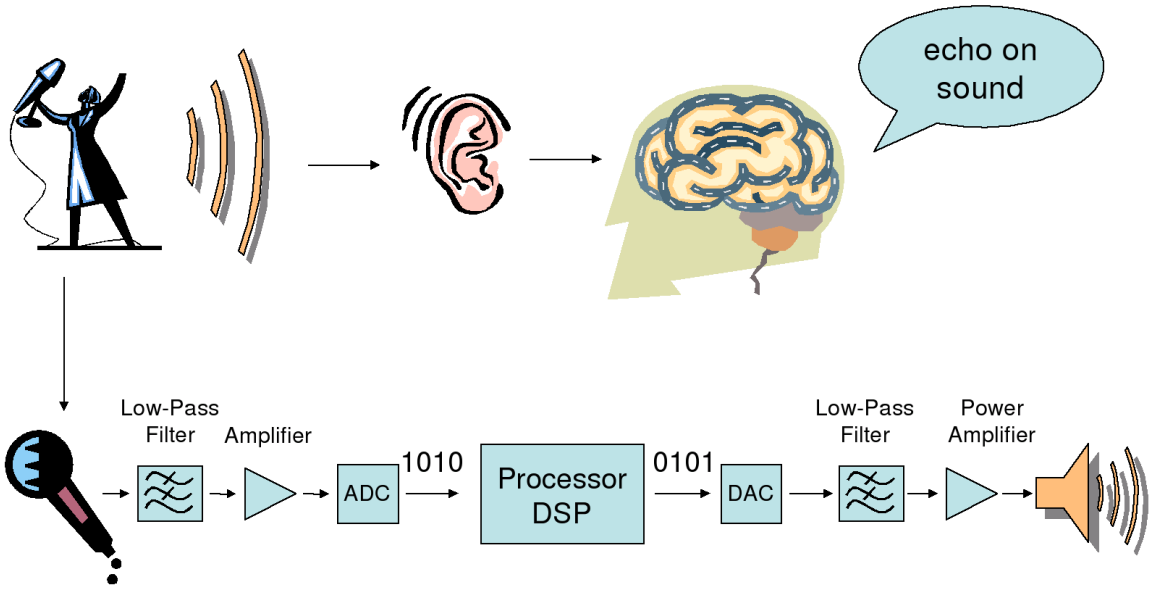

Figure 2.2: *Application: Audio digital signal processing*.

and signal bandwidth $\frac{F_s}{2}$. The digital-to-analog converter (DAC) reshapes the audio signal in the form of a normalized *sync* function, as shown in the figure. This form determines the filter response required to reject the signal components outside the required band. The output of the filter contains only the signal band of interest and some quantization noise coming from the DAC. The signal power is then increased at the output of the power amplifier and then fed to loud speakers.

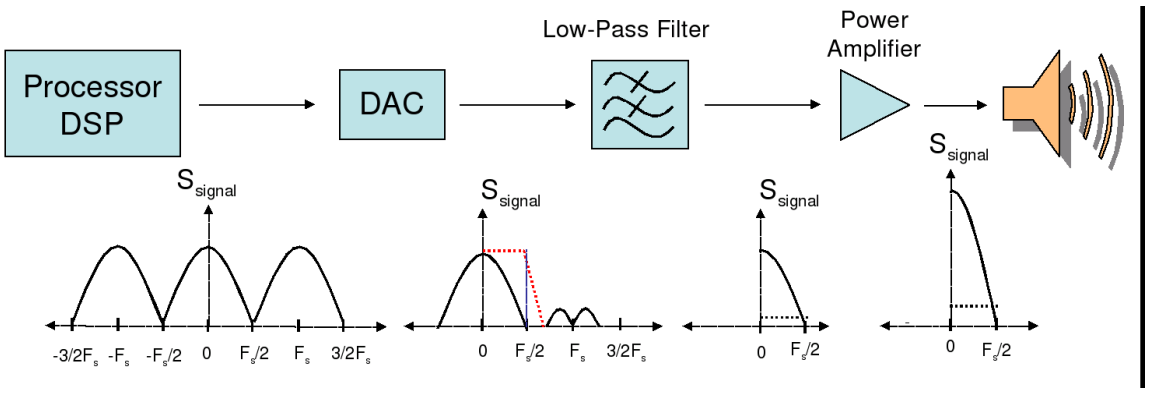

Figure 2.3: *Audio DSP: The signal spectral density at output of each block versus frequency*.

2.3.2 Filter Realization

Let us assume that the low-pass filter at the output of the DAC is to be designed. First, a filter topology is chosen to implement previously determined filter response. A designer may choose the active-RC implementation of the filter shown in Fig. 2.4. The specifications on the filter response should be translated into some specifications on the amplifier used. In general, sufficient gain and unity-gain frequency are required for the amplifier.

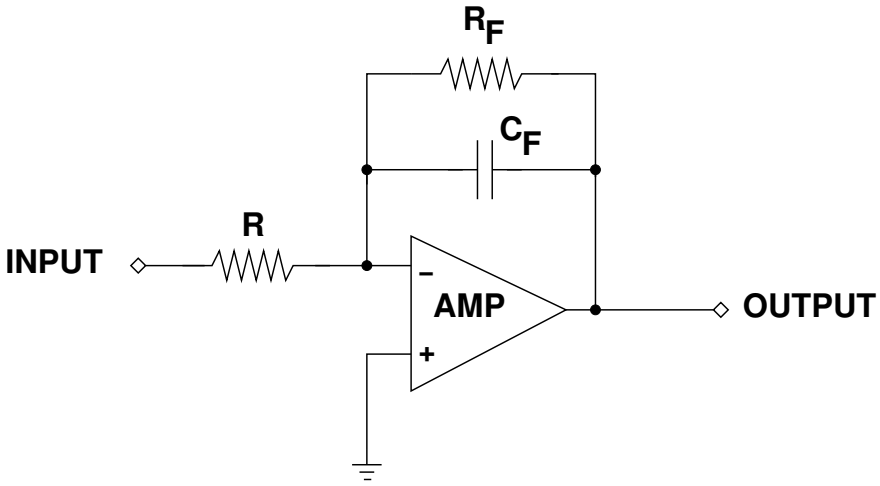

Figure 2.4: *Low-pass active-RC filter*.

2.3.3 Amplifier Realization

The definition of the amplifier specifications is shown in Fig. 2.5. We denote the static gain A_{d0}, the dominant pole F_d, the unity-gain frequency F_T and the first non-dominant pole F_{nd}.

Once the specifications on the amplifier are determined, the amplifier topology is chosen to implement them. The topology is then sized and biased using conventional analog design approaches.

For simplification, the single-stage output transconductance operational amplifier shown in Fig. 2.6 is selected. Since this topology needs to be sized, the transistor size ratios $\frac{W_1}{L_1}$, $\frac{W_2}{L_2}$, $\frac{W_3}{L_3}$, $\frac{W_4}{L_4}$ and $\frac{W_5}{L_5}$ are computed from the amplifier specifications.

Since a transistor width can reach a W_{max} of 10000μm and the physical grid size for 130nm technology is λ_{ph} is 5nm, the maximum number of possible values for the width is $\frac{W_{max}}{\lambda_{ph}} = 2 \cdot 10^6$ values per transistor. For the five transistors of the amplifier in Fig. 2.6, the total design space contains $(2 \cdot 10^6)^5 = 32 \cdot 10^{30}$ values. This design space is very huge, therefore, the designer must have efficient methods to rapidly explore this very huge design space.

Traditionally, there exist two methods to explore the design space: *simulation* and *knowledge Capture*:

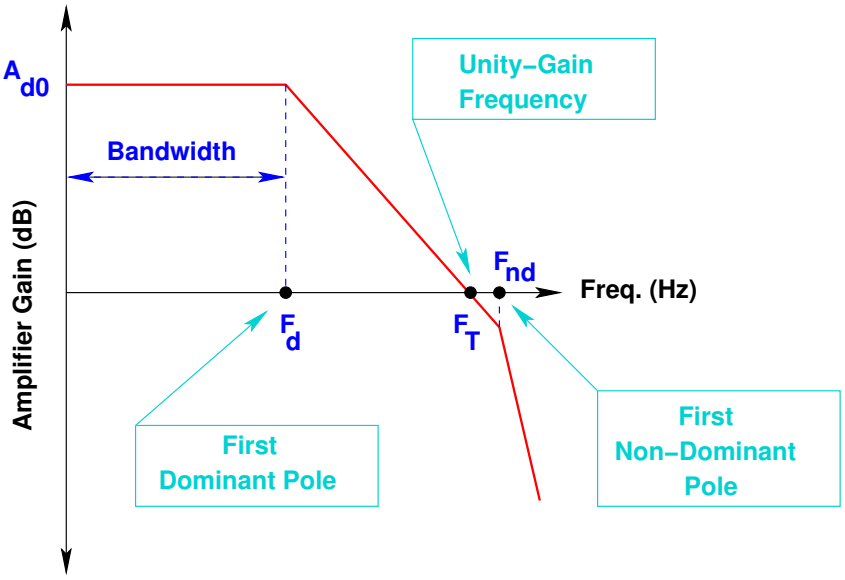

Figure 2.5: *Bode plot of the amplifier gain.*

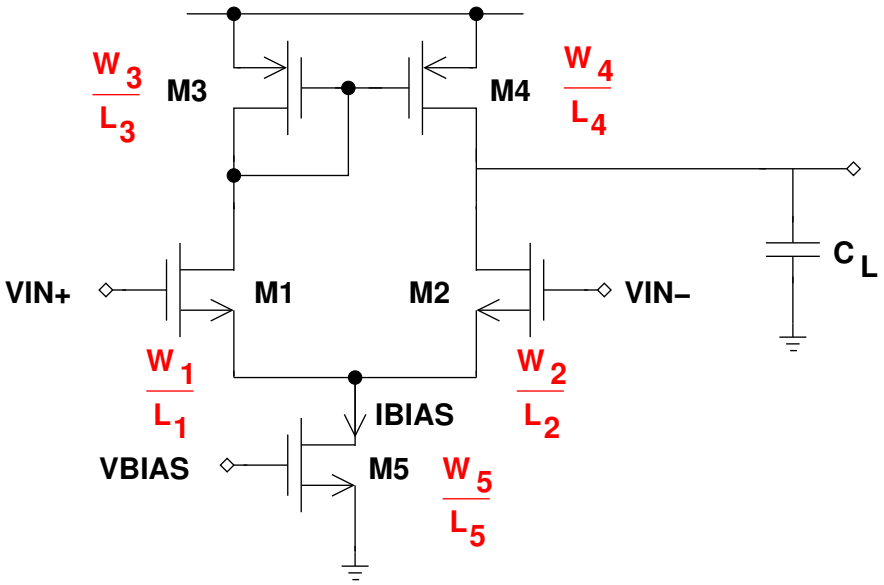

Figure 2.6: *Single stage output transconductance op-amp.*

- **Simulation:** Designs are refined by trial-and-error using an iterative time-consuming approach. This is performed using standard spice simulators. In spice simulator, circuits are described using transistor level netlists. Netlists can be *flattened* (using only transistors) or *hierarchical* (using subcircuits). Moreover, simulators use very *precise* transistor models for simulation. Examples of commercial simulators are SPICE (Berkeley Open Source), ELDO (Mentor Graphics), SPECTRE (Cadence) and HSPICE (Synopsys).

- **Knowledge Capture:** Designs are analysed by an experienced designer and equations that describe the behavior of the circuit are extracted. The method is very *time-consuming* since it requires long time to extract equations. Then it is *rapid* afterwards when directly applying them. The method is also *robust*, since the designer ensures that mismatch and other phenomena are taken into account. Since the method relies on hand calculations, designers generally use very approximate models during hand analysis. The method assumes that the circuit is flattened at the transistor level. An example tool relying on this method is OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a].

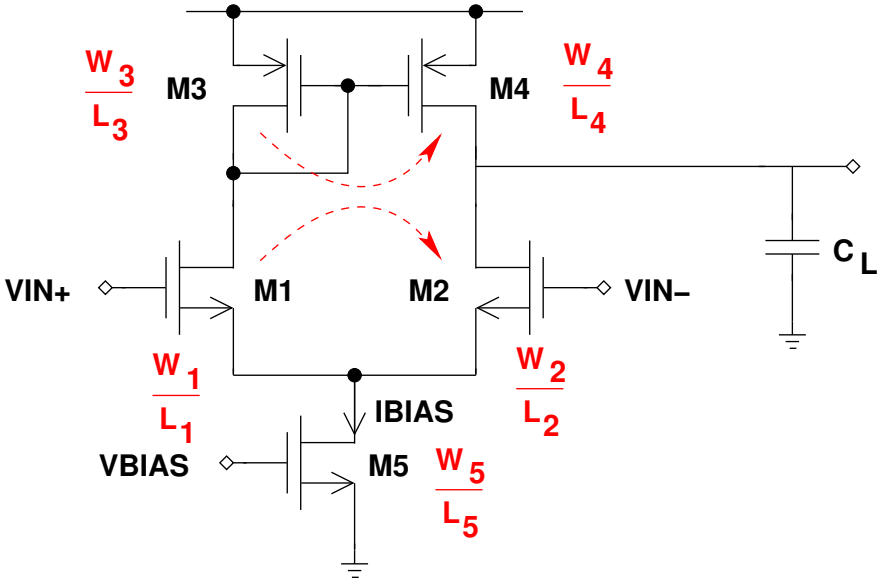

Figure 2.7: *Width propagation in OTA.*

Our main objective is to propose a methodology that combines the advantages of both methods described above. Our proposed method should be *precise* as in simulation, *robust* and *rapid* as in knowledge capture and *fully hierarchical* in order to reuse knowledge.

2.3.4 Traditional Phases of Analog Design

To illustrate the traditional phases involved into analog design, let us identify the different design steps required to size and bias the amplifier shown in Fig. 2.6. A simplified sizing procedure will be extracted out of these design steps.

Let us suppose that the width of a MOS transistor is to be computed using the quadratic model of the drain current I_{DS} in the strong inversion region,

$$I_{DS} = \frac{\mu}{2} C_{ox} \frac{W}{L} (V_{GS} - V_{th})^2 (1 + \lambda \cdot V_{DS}) \quad (2.1)$$

$$= \frac{\mu}{2} C_{ox} \frac{W}{L} (V_{eg})^2 (1 + \lambda \cdot V_{DS}) \quad (2.2)$$

where the μ is the mobility of electrons near the silicon surface, C_{ox} is the gate oxide capacitance per unit area, V_{GS} is the gate-source voltage, V_{th} is the threshold voltage, λ is the channel-length modulation coefficient, V_{DS} is the drain-source voltage and V_{eg} is the overdrive gate voltage. This very simple model is based on device physics appropriate for long-channel and uniform doping. Because the model equation is simple and easy to understand, it is still used for hand calculations and preliminary circuit simulations. However, the model has poor accuracy and scalability [Cheng99].

To simplify design steps, it is assumed that the transistors of the amplifier have equal length L. The width of the MOS transistor is then computed by inverting equation 2.1,

$$W = \frac{2 \cdot L \cdot I_{DS}}{\mu C_{ox}(V_{GS} - V_{th})^2(1 + \lambda \cdot V_{DS})} \tag{2.3}$$

This equation is used to compute the widths of M_1, M_3 and M_5. Widths are then propagated from M_1 to M_2 and from M_3 to M_4 ash shown in Fig. 2.7. Neglecting the channel-length modulation effect, equation 2.1 becomes

$$\begin{aligned} I_{DS} &= \frac{\mu}{2} C_{ox} \frac{W}{L} (V_{GS} - V_{th})^2 && (2.4) \\ &= \frac{\mu}{2} C_{ox} \frac{W}{L} (V_{eg})^2 && (2.5) \end{aligned}$$

From equation 2.4, the transconductance of the NMOS transistor is computed as:

$$\begin{aligned} g_m &= \frac{\partial I_{DS}}{\partial V_{GS}} \mid_{V_{DS}, V_{BS}} && (2.6) \\ &= \mu C_{ox} \frac{W}{L} (V_{GS} - V_{th}) && (2.7) \\ &= \mu C_{ox} \frac{W}{L} V_{eg} && (2.8) \\ &= \frac{2 I_{DS}}{V_{eg}} && (2.9) \end{aligned}$$

To compute the width from equation 2.3, the drain current I_{DS} and the transistor length L should be determined. Actually, I_{DS} should be determined from the system specifications of the filter. As explained in section 2.3.2, the filter requires an amplifier of sufficient static gain A_{d0} and sufficient unit-gain frequency F_T. Using the unity-gain frequency specification, one can determine the required current flowing in transistor M_1 using equation 2.9 in:

$$F_T = \frac{g_{m,M_1}}{2\Pi C_L} = \frac{1}{2\Pi C_L} \cdot \frac{2 I_{DS,M_1}}{V_{eg,M_1}} \tag{2.10}$$

Inverting equation 2.10, the required current flowing into M_1 is given by:

$$I_{DS,M_1} = \Pi \cdot F_T \cdot V_{eg,M_1} \cdot C_L = \frac{I_{BIAS,M_5}}{2} \tag{2.11}$$

Using the static gain A_{d0}, one can compute the required length L in equation 2.3. From small-signal analysis, the static gain is expressed by:

$$A_{d0} = \frac{g_{m,M1}}{g_{ds,M_1} + g_{ds,M_3}} = \frac{1}{V_{eg,M_1}}(\frac{1}{\frac{1}{L_1 \cdot V_{E,M_1}} + \frac{1}{L_3 \cdot V_{E,M_3}}}) \tag{2.12}$$

where V_E is the early voltage. Assuming that $L_1 = L_3 = L$ and inverting equation 2.12, one can determine the length L shared by all transistors:

$$L = A_{d0} \cdot V_{eg,M_1} \cdot (\frac{1}{V_{E,M_1}} + \frac{1}{V_{E,M_3}}) \tag{2.13}$$

From the above discussion, a first order sizing procedure for the amplifier that starts from system specifications down to performances is illustrated in Fig. 2.8. Five abstraction levels are identified for the amplifier: *system level*, *circuit level*, *transistor level*, *small-signal parameters* and *performance estimation*. At the system level, the static gain A_{d0} and unity-gain frequency F_T are first specified for the amplifier from the filter specifications. In the circuit level, these specifications are used to determine amplifier circuit parameters such as the biasing current I_{BIAS,M_5}, the differential pair current I_{DS,M_1} and the length of the transistors L. These circuit specifications are translated into the transistor level to compute the biasing voltages V_{GS,M_1}, V_{DS,M_1} and the transistor width W_{M_1}. Similar steps in the transistor level are used to compute the widths of M_3 and M_5. Once all the circuit parameters are determined, the small signal parameter g_{m,M_1} is computed the next level. Finally, the equation for the input thermal noise of the amplifier is expressed in terms of the small signal parameter g_{m,M_1}.

Since many levels of design are dealt with during the amplifier sizing, it is essential to identify the parameters that are used in each level of abstraction. The temperature $Temp$ is general parameter that should is fixed by the designer. In the system level, system parameters A_{d0} and F_T are identified. In the circuit level, circuit design parameters such as the current I_{DS,M_1} and the length L are determined out of system parameters. Note that the overdrive voltage V_{eg,M_1} is selected at the circuit level to fix the region of operation of the transistor as in the $\frac{g_m}{I_D}$ method [Silveira96]. Note also that the common-mode input voltage V_{IN+} and the common-mode output voltage V_{OUT} are generally fixed in the circuit level. In the transistor level, transistor parameters such as the biasing voltages (V_{GS,M_1},V_{DS,M_1}) and the width W_{M_1} are determined from circuit and model equations. Next, small signal parameters such as the transconductance g_{m,M_1} are determined by evaluating model equations. Finally, linear performances such as static gain A_d0, unity-gain frequency F_T, phase margin ϕ_m, ..., input-referred thermal noise $S_{th,input}$ are computed in terms of small signal parameters using performance equations extracted by experienced circuit designer. Starting from the circuit level, the parameter set at each level of abstraction and their successive transformations are illustrated in Fig. 2.9.

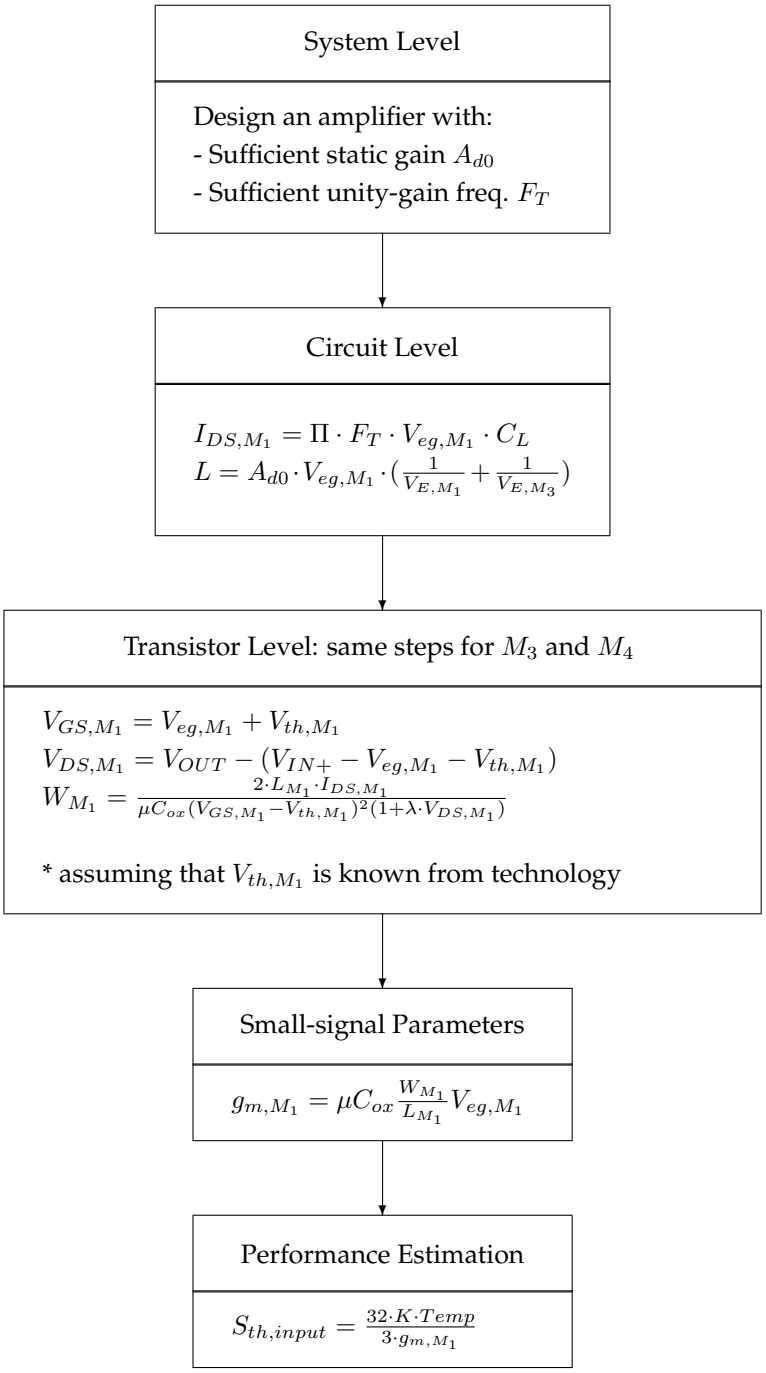

Figure 2.8: *First order sizing procedure for the amplifier*.

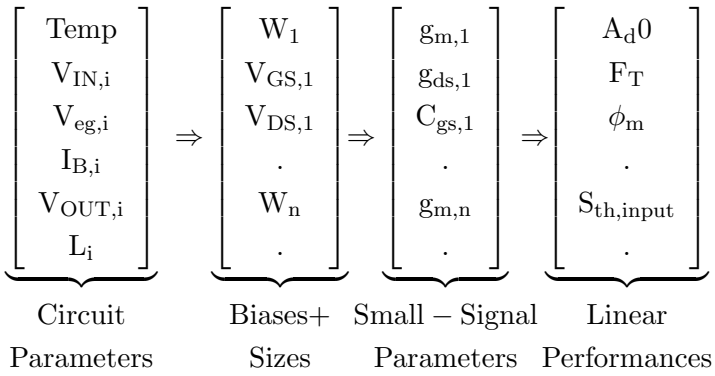

Figure 2.9: *Parameter mapping in the design space.*.

2.4 Hierarchical Sizing and Biasing Methodology

Recall from section 2.3.3 that our main objective is to propose a methodology that combines the advantages of both simulation and knowledge capture. Our proposed method should be *precise* as in simulation, *robust* and *rapid* as in knowledge capture and *fully hierarchical* in order to reuse knowledge.

These objectives can be achieved by automating the design steps, presented in Fig. 2.9, from the circuit parameters down to the small-signal parameters. The performance equations are assumed to be available through circuit hand analysis, through the use of symbolic analysis [Gielen95], or through the use of performance modeling techniques using statistical methods such Support Vector Machines [Bernardinis03], Response Surface Models [Spence99], ... etc.

In order to map the vector of circuit parameters into the vector of sizes and biases, the designer has to write a complex sizing procedure that depends mainly on the circuit topology, on designer's hypotheses, and on the different constraints that are imposed to ensure circuit functionality and robustness as in *the sizing rules method*[Graeb01]. Automating the extraction of suitable sizing procedures requires an identification of an intermediate design representation that allows us to store and analyze the sizing procedure and to convert it to well-defined consistent and top-down design steps. Introducing automation at this level, *accelerates* the design cycle of an analog circuit and ensures its *robustness*. To be *fully hierarchical*, the development of the sizing procedures should respect the hierarchy existing in the analog circuit. Respecting hierarchy encourages knowledge reuse in the basic building blocks level, in the circuit level, and in higher levels of abstraction.

One remaining factor that needs to be ensured is the *accuracy* of the sizing procedures. To map sizes and biases to the small-signal parameters, the transistor model equations need to be evaluated. Here, the accuracy is ensured by evaluating standard BSIM3V3 transistor model. Since a real model is used, the sizing procedures should attain a precision that is very comparable to simulation.

To fulfill the requirements for the hierarchical sizing and biasing, four different methodologies

have been proposed:

1. **Transistor sizing and biasing methodology:** This methodology consists of precisely sizing and biasing a transistor using standard transistor models such as BSIM3V3 Berkeley compact model. The main contribution is the development of the *sizing and biasing operators* that are considered as the interface between the transistor level and the standard transistor models. Each operator is capable of numerically inverting the BSIM3V3 model to compute unknown sizes and biases. The implementation of operators fulfills the *precision* requirement for our proposed methodology. The operators have been overloaded to support both the knowledge capture and the standard simulation approaches depicted in section 2.3.3. These are referred to by *designer* and *simulator* modes respectively.

2. **Device sizing and biasing methodology:** This methodology consists of introducing a new level called *device level* between the transistor and circuit levels. This level fulfills the requirement of hierarchy by introducing the basic building blocks that are used to construct more complex circuits at higher levels of abstraction. The methodology consists of defining an intermediate design representation, using this representation to automatically create suitable sizing procedures for each building block, and storing the sizing procedure in the corresponding building block for later reuse. The sizing procedure respects the constraints imposed on the building blocks to ensure their correct functionality. This fulfills the requirement of *robustness* for our proposed method.

3. **Circuit sizing and biasing methodology:** This methodology is the most laborious task. It targets the circuit level and consists of defining circuit structure as an interconnection of basic building block from the device level. The methodology combines the sizing procedures of building blocks in order to create more complex sizing procedures for the whole circuit. This way knowledge stored in the building blocks can be reused in higher levels of abstraction. This fulfills both requirements on *hierarchy* and *knowledge reuse*. The methodology also analyzes the knowledge presented in the sizing procedures and detects inconsistencies such as incomplete knowledge, or conflicting hypotheses. It defines repairing mechanisms to correct those inconsistencies and transform the sizing procedure into a consistent and reusable top-down knowledge.

4. **System Level methodology:** This methodology targets levels of abstraction higher than the circuit level. These levels of abstraction are simply knowledge that enriches the design representation of the circuit level. This enrichment may be in the form of constraints at the system level or system equations that are stored in executable procedures. Unfortunately, this methodology will be not be tackled in the present work.

The automatic extraction of suitable sizing procedures and their use in the computation of DC operating point is made possible through the use of a design representation called *dependency*

graphs. Through the use of the dependency graphs, each level of abstraction can add knowledge to enrich the dependency graph from the transistor level up to the system level. This way the whole system knowledge can be stored in a reusable form and further analyzed to ensure its consistency. Therefore, the *the hierarchical sizing and biasing methodology* is proposed to fulfill the above requirements. Applying this methodology, one anticipates that the resulting design representation contains all the required knowledge needed for sizing from the system level down to the transistor level. The knowledge contribution of each level of abstraction is clearly documented in the design representation. The design representation itself is a reusable and executable knowledge that can be easily analyzed. This gives lots of knowledge insight to design automation tools.

We summarize the objectives of the hierarchical sizing and biasing methodology as follows:

1. The nature of analog design knowledge is understood by design automation tools.
2. Knowledge is represented in a unified intermediate design representation. Hence, knowledge reuse is capitalized.
3. The choice of the circuit design parameters in Fig. 2.9 are intuitive to the designer. Instead of dealing with widths, the designer manipulates currents and voltages.
4. Suitable sizing procedures are automatically extracted from the circuit topology.
5. The methodology is precise as in simulation approach.
6. The methodology is fully hierarchical since hierarchy is explicitly defined and used.
7. The methodology is rapid since it accelerates the design cycle of demonstrated analog circuits.
8. The methodology is robust since the sizing procedures respect constraints and hypotheses expressed by the designer.
9. The methodology supports both knowledge capture and standard simulation approaches through the *designer* and simulator modes.

2.5 Proposed Tool Architecture

The use of our proposed methodology inside an optimization loop proved to reduce the design space during optimization. This is due to the fact that the circuit design parameters chosen in the first vector of Fig. 2.9 have narrow ranges of variations compared to the widths when used as optimization variables [Krasnicki99]. To demonstrate the effectiveness of the hierarchical sizing and biasing methodology, a tool architecture has been proposed as shown in Fig. 2.10. The design tool is an automated knowledge-based synthesis tool. Based on the selected set of circuit parameters, the synthesis system automatically generates suitable sizing procedures for the analog IP.

The analog IP is described starting from circuit level down to performances. Once the sizing procedure is generated, it is executed to computed the sizes and biases for every transistor. Then, the models are evaluated to determine the small signal parameters. These are later used to evaluate performance equations supplied by the designer. The optimal values of design parameters are searched for using a search engine. The search engine uses the *Nelder-Mead Simplex* [Nelder65] method.

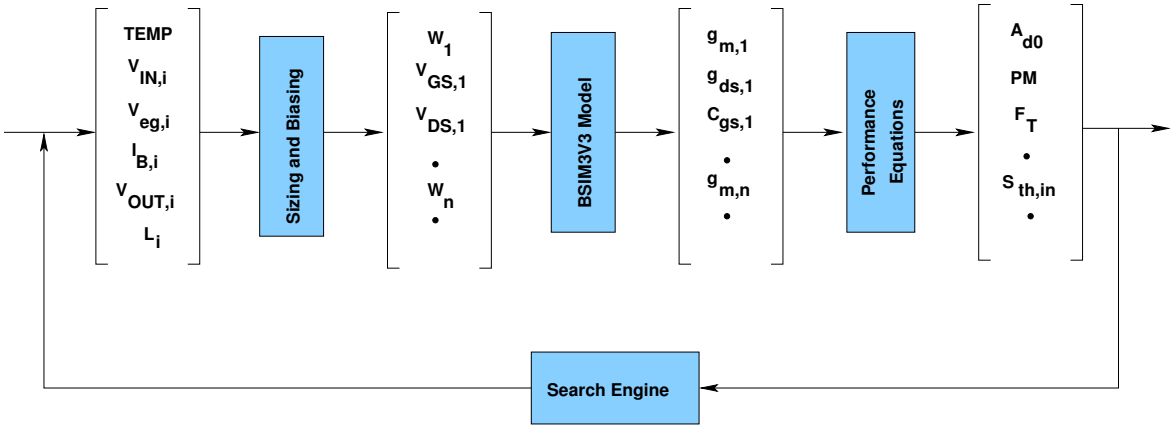

Figure 2.10: *Block diagram of the proposed synthesis system.*

The whole synthesis system was introduced in our analog design reuse environment called *CAIRO+*. Now, CAIRO+ supports both the *standard* and *automated* modes. In the standard mode, the designer documents his design knowledge in the form of reusable firm IP. In this mode, the designer himself must ensure the consistency of knowledge. In the automated mode, the knowledge is automatically extracted from the circuit topology and its consistency is automatically ensured by the synthesis system.

2.6 Conclusions

To keep pace with rapidly evolving markets, analog intellectual properties (IP) has to be widely adopted. This requires the identification of a canonical format to describe IP cores. One such used format is the firm IP. Firm IP blocks are provided as parameterized circuit descriptions so that designers can optimize cores for their specific design needs. Firm IPs appear to be the most appropriate format to deliver AMS IP library components. Today, analog firm IPs are hardly synthesized from specifications to layout. Therefore, advanced design tools and methodologies are needed to explore the potential of firm IPs.

In this perspective, we have presented a design example of an audio DSP processor. Throughout the example, we identified requirements for methodologies targeting the synthesis of firm IPs. From these requirements, we propose a methodology for hierarchically sizing and biasing firm

IPs, along with a tool architecture to evaluate it. The following chapters describe in details of this methodology.

Chapter 3

State of the Art

3.1 Introduction

Analog design automation is a very complex task. An efficient design of an analog intellectual property touches many levels of design abstraction at the same time, namely *system level*, *block level*, *circuit level* and *transistor level* [Jancke06, Doboli03, Martens08]. For this purpose, this chapter introduces the state-of-art of the different fields that have direct impact on the development of the hierarchical sizing and biasing methodology.

Since our proposed methodology computes the DC operating point, section 3.2 presents the *methods of DC operating point computation*. Three different paradigms are compared, namely: *standard simulation*, *relaxed DC formulation* and *operating point driven formulation*.

In our proposed methodology, the DC operating point computation reposes on the integration of a compact transistor model. Therefore, we introduce in section 3.3 and section 3.4 two important fields, namely: *compact device modeling* and *Model development and standardization efforts* that have largely occupied the industry and academia. Since the industrial aim is to integrate models that accurately describe the behavior of physical devices, efforts for modeling and standardization are inevitable to ensure the spread of compact models in the wide electronic community.

Since our proposed methodology is to be used for both synthesis and reuse for analog cells, we present, in section 3.5, the state-of-art knowledge-based synthesis and design reuse methodologies that have been developed in the last few decades.

Since our aim is to develop an intermediate design representation for analog, we present in section 3.6, the known state-of-art design representations.

Finally, section 3.7 concludes the chapter.

3.2 Methods of DC Operating Point Computation

A very important topic to investigate in both approaches, is the DC operating point (DCOP) computation. As DCOP computation is an indispensable task in knowledge-based synthesis, it is very

critical for simulation-based synthesis. In the latter approach, a simulator is called to compute the DCOP at each design point in the design space. DCOP computation [Pillage95] is performed by solving for node voltages using *Newton-Raphson* iterations. In each iteration, the circuit is linearized and the admittance matrix is formed using *Modified Nodal Analysis* [Ho83] technique. Then the admittance matrix is inverted to get an estimate of the node voltage, using either *LU decomposition* or *Gaussian elimination*. This estimate is introduced into Newton-Raphson equation to compute a better estimate for the next iteration. Note that one Newton-Raphson iteration requires the evaluation of the Jacobian matrix for the nonlinear function that is solved. This process continues until Newton-Raphson converges. In addition to these steps, some runtime overhead may exist during the simulator execution and control. These factors contribute to the increase of execution time of individual simulations. The impact on execution time will be great, as discussed in MAELSTROM [Krasnicki99], that executes thousands of simulations during optimization. It is clear that DC operating point computation is a mandatory task for both knowledge-based and simulation-based synthesis. In the following, we describe several approaches for the DC operating point computation.

3.2.1 Standard Simulation

In a simulator like SPICE, the DC solution is obtained by solving a system of simultaneous nonlinear equations with a Newton-Raphson root solver. The independent variables are the node voltages $V_n = [V_1, \cdots, V_N]$. The number of independent variables of the root solver is thus N. Given the node voltages V_n, one can calculate the branch voltages $V_b = A^T \cdot V_n$, with A the incidence matrix [Vlach94]. The branch constitutive equations $g(V_b)$ allow to determine the device currents out of the branch voltages and the device dimensions. The constraints are the Kirchhoff's Current Law (KCL) of the different nodes. Given the branch currents, one can calculate the error on the KCL's. The root solver has to solve the system of simultaneous nonlinear equations:

$$V_b = A_T \cdot V_n \quad \rightarrow \quad I_b = g(V_b) \quad \rightarrow \quad \delta_{KCL}(A \cdot I_b) = A \cdot I_b - J_n = 0 \tag{3.1}$$

where δ_{KCL} is the error on the KCL and J_n is the nodal current source vector of the nodal formulation. The above root solving method is an incomplete solution method: one can't guarantee that the solution will always be found. The iterative Newton-Raphson scheme can fail to converge.

3.2.2 Relaxed DC Formulation

Several researchers attempted to solve the problem of DCOP computation during synthesis. Maulik [Maulik93] used a relaxed DC formulation that specifies Kirchhoff's Voltage Law (KVL), Kirchhoff's Current Law (KCL) and performance goals as constraint functions. Then, the resulting constrained optimization problem is solved using sequential quadratic programming techniques (SQP). In a relaxed DC formulation, the node voltages V_n and the device dimensions

W and L of the MOS transistors are the independent optimization variables. Similarly, in ASTRX/OBLX [Ochotta96], all KCL and KVL equations were presented as penalty terms in the overall cost function.

This has two major drawbacks. With a design space scattered full of local minima [Ochotta96], efficient local optimization is excluded. It becomes necessary to do the optimization with a computationally expensive global optimizer such as simulated annealing. The possible speed gain obtained by not pursuing an exact DC solution during each optimization iteration is lost by this.

A second drawback of the introduced local optima is that DC consistency is less guaranteed. DC consistency requires that one achieves the real global optimum. This requires long annealing times. Avoiding DC inconsistency by increasing the weights on the KCL error measures is tempting but results in a premature freezing of the node voltages. These are key design parameters which should be frozen only at the latest moment. As a consequence, the situation may occur frequently that optimization have to be run multiple times because the obtained results are DC inconsistent.

3.2.3 Operating Point Driven

In operating point driven sizing[Plas01, Leyn98], one specifies the operating point and determines the device dimensions W out of it. The independent variables are the node voltages V_n, a set of independent chord currents I_c and the L's. Out of the node voltages V_n, the branch voltages $V_b = A_T \cdot V_n$ are determined, and out of the chosen set of independent chord currents, the branch currents I_b are determined $I_b = B_T \cdot I_c$ with B the basic loopset matrix [Vlach94]. Given the L's of the devices, the W's of the devices can be determined since the $I_{DS,b}$ currents are branch currents that are calculated out of the chord currents:

$$I_b = B_T \cdot I_c \tag{3.2}$$

$$\delta_{W_i}(W_i) = I_{DS,b} - I_{DS,i}(V_{GS,i}, V_{DS,i}, V_{BS,i}, W_i, L_i) = 0 \qquad i = 1, \cdots, M \tag{3.3}$$

This is a sequence of M one-dimensional problems. For each transistor, one solves $W = W(I_{DS}, V_{GS}, V_{DS}, V_{BS}, L)$. The root solving is always converging since the function $W(I_{DS}, V_{GS}, V_{DS}, V_{BS}, L)$ is monotonic and thus can be solved with a bisection method. A solution is thus guaranteed (even if it is unfeasible i.e. $W < W_{MIN}$), making it a complete method.

3.3 Compact Device Modeling

Compact device modeling [Brooks99] is defined as the process of developing device model equations used for the electrical representation of the physical behavior of a device. The word *compact* is used because these equations are simplified based upon several assumptions that are made when developing the model equations. Many of the equations in today's compact models would hardly

seem brief or simplified; however, assumptions are still needed to arrive at unique solutions that can be used by SPICE programs.

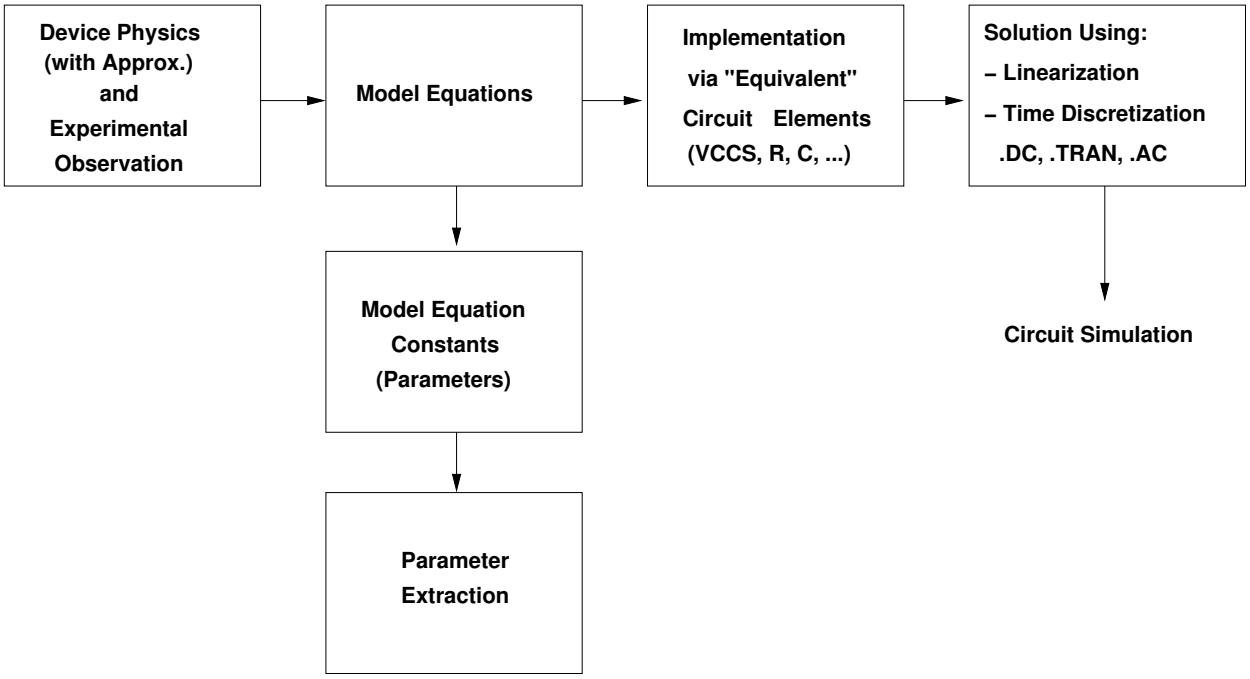

Figure 3.1: *Development cycle of a compact model*.

Fig. 3.1 shows the development cycle of a compact model. A model developer uses device physics with some approximations (using experimental observation) and develops a set of model equations. The model equations have associated constants (parameters) that need to be determined from parameter extraction. Once a complete set of these parameters is established, the model is then implemented into the SPICE programs. SPICE represents these equations as equivalent circuit elements (voltage controlled current sources, resistors, capacitors, etc.) and then solves the equations based upon the type of simulation requested. SPICE's engine is a linear equation solver. Therefore, all the nonlinear elements must be linearized using Taylor series expansion and then solved via iterative techniques.

Currently, two of the fastest growing segment of the industry are fabless company and the foundry [Foty97]. The fabless company relies on simulation of their design without having the opportunity to build it first. The foundry puts the design into the integrated circuit without doing the design environment. The only means of consistent communication regarding the design and its performance is the electrical compact model.

In our proposed methodology, we mainly focus on the integration of the standard compact model *BSIM3V3* (BSIM for Berkeley Short-Channel IGFET Model) [Cheng99] developed at the University of California, Berkeley, CA. The model was selected as the first MOSFET model for standardization by the *Compact Model Council* (CMC).

3.4 Model Development and Standardization Efforts

As the semiconductor industry grew and new technologies were introduced, a need for electrical models that SPICE could use for simulation grew accordingly. Many companies addressed this need by creating focused model-development groups with the results being used internally as proprietary models. Companies that did not have the luxury of investing many man years of effort into model development depended on the models inherent in SPICE programs. Unfortunately as technology progressed, models lagged behind in accuracy and in availability.

To face this lag in model development, the CMC was chartered to promote the international, non exclusive standardization of compact model formulations and model development. The CMC consists of 36 leading members in the semiconductor industry such as Cadence Design Systems, Compaq, Conexant Systems, Hewlett Packard, Hitachi, IBM, Intel, Lucent Technologies, Mentor Graphics, Motorola, Siemens, and many others. The CMC chose the BSIM3V3 model as the first formally standardized compact model. Since BSIM3V3 is a very accurate, scalable, low-voltage, high-speed, analog/digital MOSFET model, it has allowed CMC to develop a methodology for standardizing compact models. The initial task was to develop quantitative and qualitative tests for compact models. Tests were developed to show the "goodness" of a compact model [SIA98]. These tests were not designed to show how accurate a model was, rather to document the shortcomings of the model. These shortcomings can cause simulation problems such as convergence errors, inaccuracies in the simulation results, or longer than necessary runtime for the simulation. The model equations by themselves could not be tested without a parameter set, so an initial extraction methodology was also required. The standardization tasks included improved code robustness, bug fixes, improved documentation, correction of numerical errors, inclusion of more accurate portrayal of physical phenomenon, and software management procedures including revision control and release procedures. Today, the CMC supports the standard models shown in Table 3.1. The CMC also supports models for passive devices such resistors and varactors[Watts06].

Compact Model	Developed at	**Model Type**
PSP	Penn State-Philips	MOSFET Transistor
HICUM	Dresden University of Technology, University of California San Diego	Bipolar Transistor
MEXTRAM	Philips	Bipolar Transistor
BSIMSOI	IBM	SOI CMOS
LDMOSFET	Delft University of Technology	High-Voltage MOSFET
BSIM3	University of California Berkeley	MOSFET Transistor
BSIM4	University of California Berkeley	MOSFET Transistor

Table 3.1: *Compact Transistor models supported by CMC [Watts06].*

In parallel with the early work on BSIM3, the CMC also worked on establishing an indus-

try standard model to simulator Application Program Interface (API). Having such an interface would greatly speed model development by simplifying the process of installing a model in a simulator where it can be tested with a variety of circuits. Many attempts have been done to build model compilers that reads compact device models described in high-level languages such as VHDL-AMS and Verilog-AMS and automatically generate the simulator device code in C that can be directly linked with existing circuit simulators such as SPICE3. For example, MCAST [Wan03, Hu05] shown in Fig. 3.2 reported the successful implementation of industry grade device models, including EKV, BSIM and BSIM-SOI, in VHDL-AMS and Verilog-AMS.

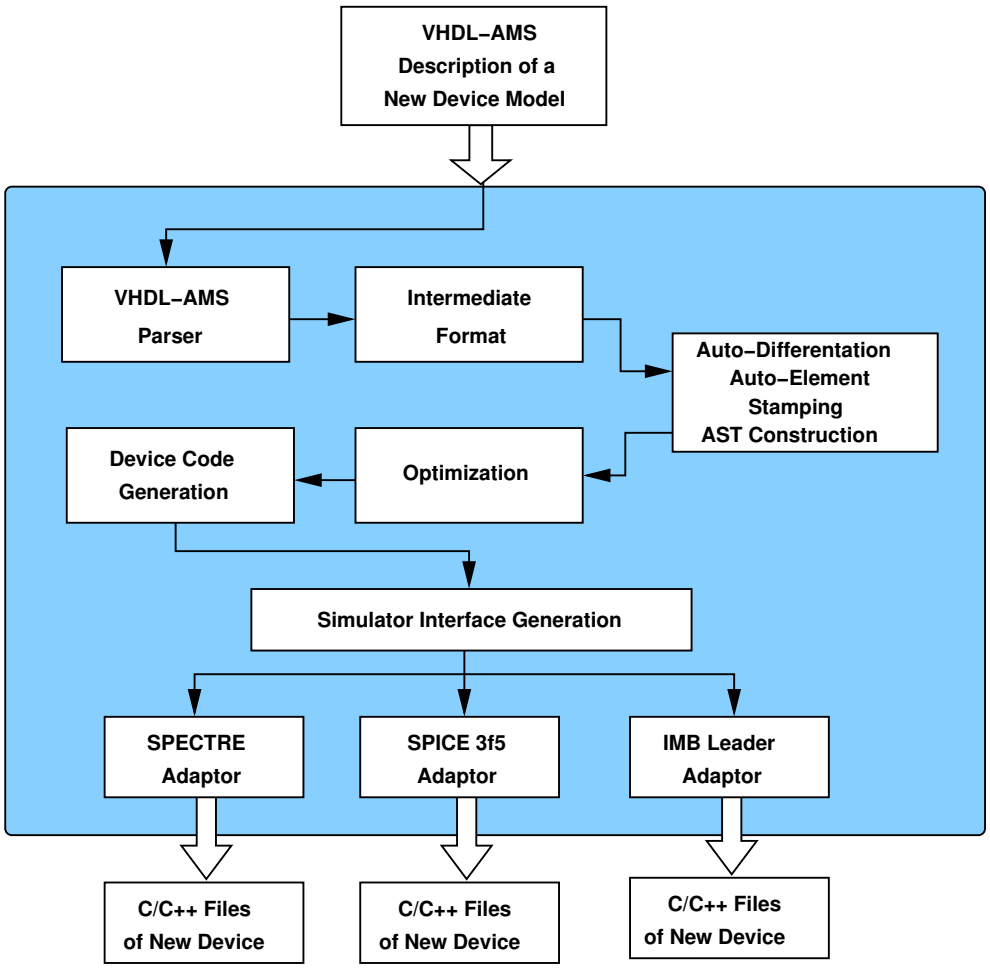

Figure 3.2: *MCAST model compiler architecture [Wan03, Hu05].*

Commercially, some interesting methodologies have been proposed. For instance, in Mentor Graphics Mixed-Mode Simulator ELDO^{TM}, the *user-defined modeling language* (UDM) and *the generalized user-defined modeling language* (GUDM) have been proposed. UDM (User-defined models) is an interface which allows the user to write their own model. UDM can be used to implement BJT, Diode, JFET and MOSFET. The only limitation of UDM is that the equivalent circuit of the

model to implement is imposed, and cannot be changed. For instance, a new diode model can be implemented via UDM only if the model returns only one charge, one static current, both between the intrinsic anode and the cathode, an access resistor being optional. If the user wishes to implement a diode model with two internal nodes, UDM cannot be used, rather GUDM is used. GUDM is a more general interface which allows implementation of devices with varying equivalent circuit. The internal equivalent circuit of the model is not restricted, the model can have an unlimited number of internal nodes, static currents, and charges. The only limitation is that the number of external pins be the same as the number of pins of the generic model (e.g. BJT: 4 pins; diode: 2 pins). To implement a user model, a set of functions, used as initializations, must be filled out and a new one, used for the model itself, must be coded from scratch. Once implemented, the UDM/GUDM model is compiled and linked to ELDO^{TM} executable file. The model can be debugged using a simple UNIX debugger.

A similar commercial product called ModelLib^{TM} is proposed by Simucad [Simucad]. The Simucad SPICE Model Library (ModelLib), shown in Fig. 3.3 is a collection of all of the models delivered with Simucad circuit simulators as a dynamically linked library. Individual models are distributed online as pre-compiled, pre-linked, pre-tested binary models that simulator users can easily download and install when a required model update is available. ModelLib enables access to the most up-to-date high performance SPICE models for:

1. **SmartSpice** Analog Circuit Simulator

2. **SmartSpice-RF** Harmonic Balance Based RF Simulator

3. **Harmony** Analog/Mixed Signal Simulator

4. **Twister** Full-Chip Hierarchical Analog Circuit Simulator

Simucad supports MOSFET models, SOI models, DIODE models, TFT models, BJT models, MESFET models, JFET models, FRAM models and HBT models.

Simucad offers also the model library development environment shown in Fig. 3.4. It allows convenient and fully independent environment for proprietary model development. Model source codes and binaries may be shared among different model development teams. It allows easy and rapid generation and run of a complete set of regression tests for an independent and isolated model.

Another important initiative in Europe, is the MOS *Modeling and Parameter Extraction Group* (MOS-AK) [MOS-AK] which aims to encourage interaction and sharing of all information related to the compact modeling at all levels of the device and circuit characterization, modeling and simulations. One of its main targets is to promote standardization of the compact models and its implementation into software tools.

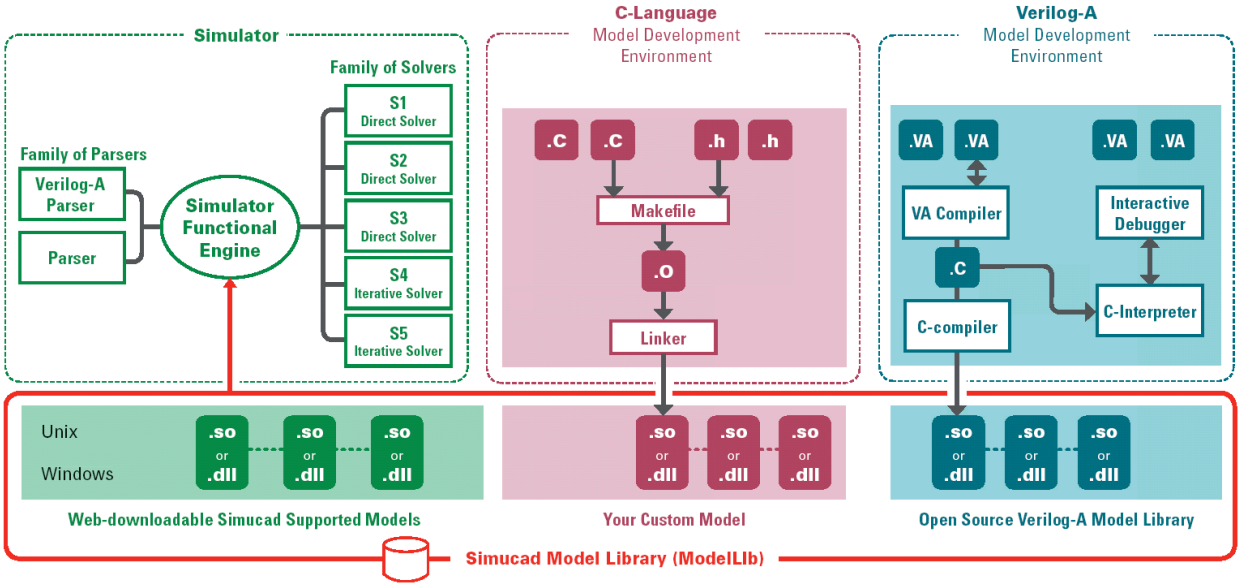

Figure 3.3: *SimuCAD: ModelLib Dynamically-Linked SPICE Models [Simucad].*

3.5 Analog IP and Design Reuse

3.5.1 Optimization-Based Synthesis Tools

A firm IP can be reused by resynthesizing it using optimization-based synthesis tools. These are classified into two distinct flows: simulation-based and knowledge-based. A simulation-based synthesis flow, such as MAELSTROM [Krasnicki99], runs an optimizer in a closed loop with an analog simulator. Both communicate circularly till an optimal set of values is achieved for design parameters and the specifications are met. This is shown in Fig. 3.5.

The simulation-based approach uses standard commercial transistor models. Therefore, sizing accuracy is always ensured. As distinct from the simulation-based approach, a knowledge-based synthesis flow is normally based on approximate transistor models. In this approach, the designer has to identify all the circuit equations describing circuit performances in terms of design parameters. The designer must also identify a *design plan*, i.e. an appropriate sizing procedure which is based on circuit equations, in order to compute sizes and biases. Once determined, the design plan can be coded using a programming language as C or MATLAB in the form of a reusable knowledge. In OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a], the codification process includes the computation of DC operating point and the dimensions for each sub-block used in the hierarchy of the analog circuit.

The knowledge-based approach is more appealing to the designer than the simulation-based one. Designers may not trust simulation-based synthesis results or may consider it incomplete.

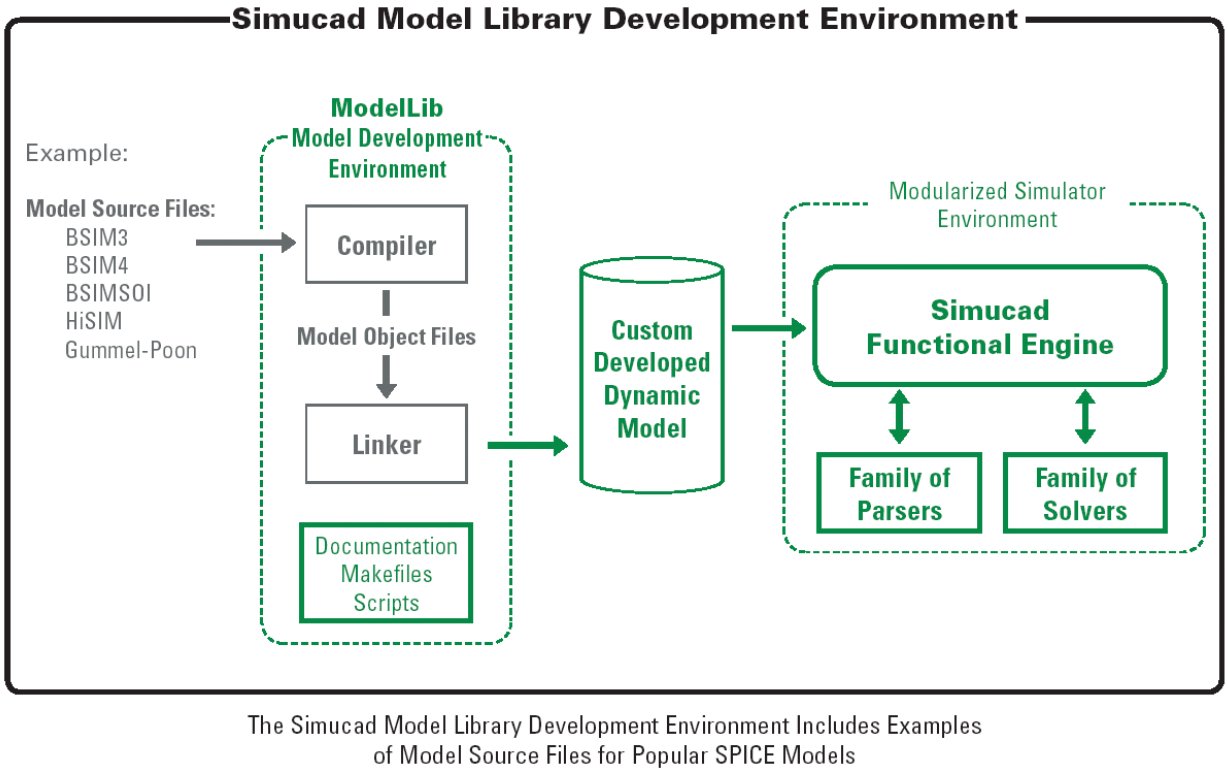

Figure 3.4: *The Simucad Model Library Development Environment [Simucad].*

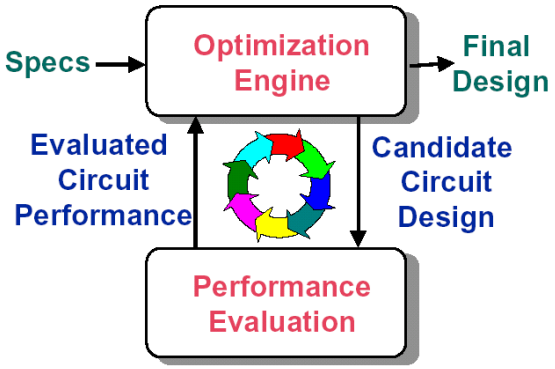

Figure 3.5: *Simulation-based optimization.*

Therefore, to resolve the problem of approximate models, knowledge-based synthesis systems started to incorporate accurate models borrowed from analog simulators. Moreover, the selection of design parameters in simulation-based synthesis systems in not so obvious. Most systems directly optimize the dimensions of transistors while many designers tend to use characteristic currents and voltages. In knowledge-based systems, the designer has full control over the choice

of parameters.

We mainly focus our study on knowledge-based synthesis tools. Many research attempts regarding this class of synthesis tools have been reported in literature. Those are namely: OPASYN [Koh87, Koh90], OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a, Carley90], IDAC [DeGrauwe84b, DeGrauwe84a, DeGrauwe87b, DeGrauwe87a, Degrauwe87c, DeGrauwe89], ARIADNE [Swings91a, Gielen89, Gielen90b, Gielen90a, Gielen91, Swings91b, Wambacq91], STAIC [Harvey92], ISAID [Makris92, Makris95, Toumazou90], Maulik [Maulik93, Maulik91, Maulik92b, Maulik92a, Carley93] and AMGIE [Plas01]. Table 3.2 compared in details the reported state-of-art knowledge-based synthesis tools.

3.5.2 Firm IP Hardening Flow

The IP hardening flow has been discussed in section 2.2. Fig. 3.6 shows a more detailed flow for hardening a firm IP using its different views: *schematic*, *analytic*, *testbenches* and *physical*.

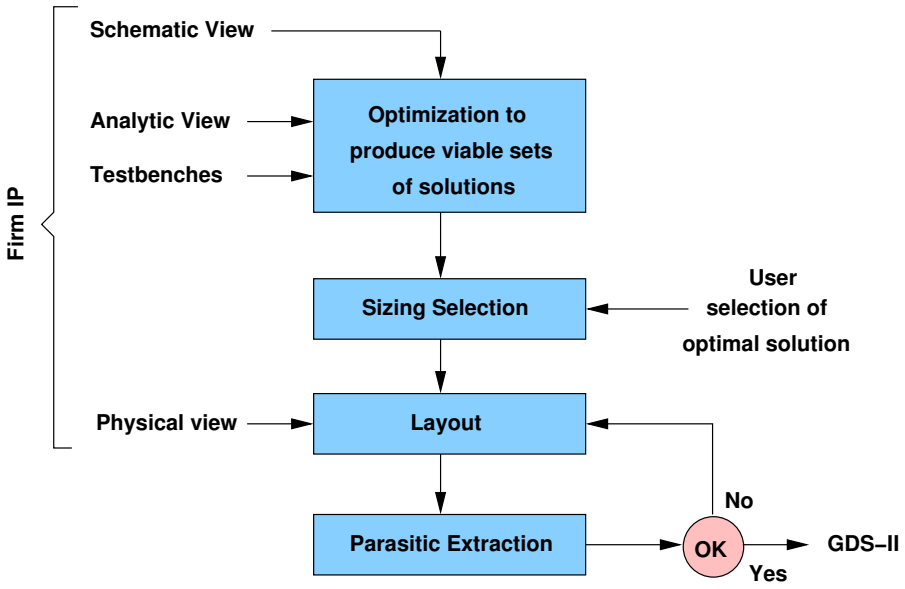

Figure 3.6: *Analog/mixed-signal firm IP hardening [Hamour03]* .

The concept of IP hardening [Hamour03] can be generically defined as the process of working from a firm IP and a set of specifications to the production of the GDS-II layout. First an architecture/topology is selected using the behavioral and analytical models associated with the IP blocks; the selected blocks must satisfy the specifications supplied by the user. Once a topology is selected, the firm IP is optimized by simulating the testbenches to measure performance; the circuit is optimized over process corners with typical input/output loading effects included. Next, the circuit layout is performed; this is followed by parasitic extraction and re-simulation; adjustments to the layout are performed until the specifications are satisfied. For a physical synthesis

flow to be effective, it must have the knowledge of the typical layout structure for this type of block.

3.5.3 Scaling Rules

The migration of an analog circuit to a new technology usually requires a complete redesign. Several researches attempted to calculate parameters for a given circuit in a new technology, starting from the same design in an earlier technology.

Galup *et al*[Galup-Montoro00, Galup-Montoro02] studied the case of maintaining the original dynamic range and the gain-bandwidth-product specification using two different reuse strategies: *constant-inversion-level scaling* and *channel-length scaling*. Constant-inversion-level scaling keeps $\frac{g_m}{I_D}$ constant while channel-length scaling take advantage of the smaller dimensions of a new generation technology. The most important drawback of this method is the increase of the chip area during a decrease of supply voltage. Hence, the area of the migrated circuit will be bigger than the original area. This is due to the dependence of the product $W \cdot L$ on technology parameters only. These cause the scaling rule for the product $W \cdot L$ to be greater than 1.

Acosta *et al* [Acosta02] studied the impact the redesign method presented in [Galup-Montoro00, Galup-Montoro02] on two additional performance aspects (slew rate and current mirror frequency response). The results show that the method decreases the slew rate and hence this is an aspect to look after in the resulting design. Regarding the current mirror frequency response, Acosta's analysis shows that if the current mirror transistors were originally designed to work in stronger inversion than the differential pair transistors, the current mirror pole frequency increases, preserving a good overall frequency response. In addition, the possibility of applying the bias current as a tuning parameter to customize an existing design for different applications was analyzed. The results confirm that this approach works well for weak and moderate inversion designs.

Savio *et al* [Savio06, Savio04] proposed an enhanced method based on [Galup-Montoro00, Galup-Montoro02] to obtain a scaled circuit with a frequency behavior similar to the original one. A tuning procedure modifies the transistor aspect ratios in order to properly scale the transistor's transconductances and the output conductances. When transistor's parasitics reduce the frequency performances, an additional numerical loop is performed. In this loop, a Levenberg-Marquadt optimization procedure is applied until the distance between the two curves becomes lower than a threshold. Since the transconductance fitting may not ensure that the transient response are correct, a fitting procedure similar to the frequency fitting is applied to the transient response. To conclude, Savio's method improves the original one [Galup-Montoro00, Galup-Montoro02] since the DC gain is preserved and the power consumption is decreased. However, the area drawback still exists.

The analog design reuse method proposed by Levi [Levi07a] solves the problem of area. The proposed method allows the designer to choose the strategy of redesign, hence, the equations

which will be used to determine the scaling factors. The choice of strategy depends on the designer goals and system specifications. The method does not suffer from the problem of area: the chip area decreases with newer technologies. Parasitic effects are not taken into consideration. Solutions similar to Savio's work [Savio06, Savio04] could compensate parasitic effects.

3.5.4 IP-Based Library

In the approach presented by Levi[Levi07b], a database holds the hierarchy of analog IP blocks and their contents at different levels namely *system*, *macro-block*, *block*, *cell* and *transistor* levels. Each IP block is described by different views: *symbol*, *connectical*, *functional*, *behavioural*, *schematic*, *layout* and *characterization* views. The objective of the exploration phase is to find one ASIC solution according to the initial specifications. The chosen method is to perform a top-down exploration, from the *macro-block* level to the *cell* level using validity domain as a selection criterion. If many different corresponding IPs are found at one level, the one with the largest validity domain is selected. The method was applied to a neuromorphic ASIC as shown in Fig. 3.7.

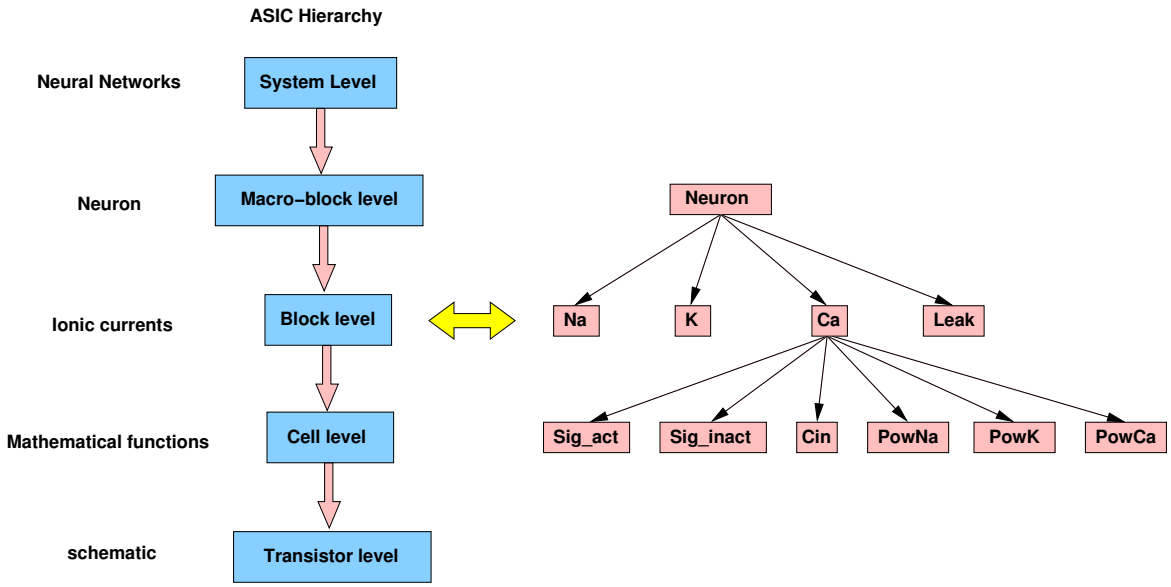

Figure 3.7: *IP-based library for a neuromorphic ASIC [Levi07b].*

3.5.5 Template-Based Layout Retargeting

In analog design, circuit functionality and performance are extremely sensitive to physical implementation details. Therefore, it is critical that a layout be optimally implemented. This begins with a well-chosen topology or architecture followed by carefully crafting all physical details. Since reaching optimal circuit behavior is an incremental task, designers need to have complete control at each step of the physical implementation thereby enabling predictable results.

A simple and proven methodology to achieve these goals is to use well-designed templates for each type of circuit. Such a layout template includes all the circuit devices and objects and the connectivity.

The relative placement and orientation of these elements have been carefully chosen in the template. The properties that are going to be modified for each derivative implementation are the actual device parameters, sizes and wire dimensions. The template is flexible enough to accommodate variations in device parameters and overall cell dimensions but maintains the basic layout characteristics that were chosen for this family of circuits.

This approach is adopted by Sagantec to migrate circuit layouts for hard IP [Sagantec] as depicted in Fig. 3.8.

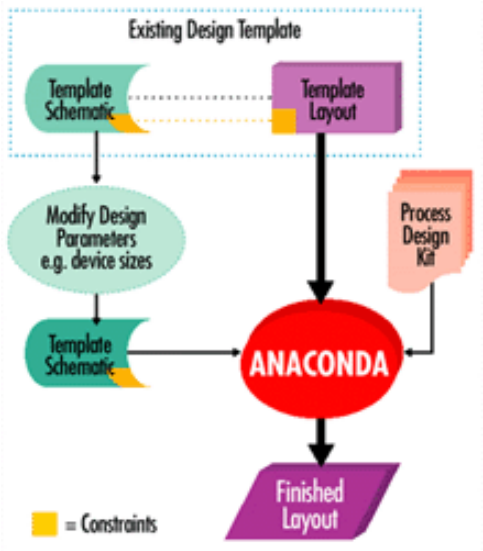

Figure 3.8: *Sagantec template-based layout retargeting [Sagantec].*

3.5.6 Recent Knowledge-Based Synthesis Tools

3.5.6.1 CAIRO+: Creating Analog IPs - Reusable and Optimized

Over the last 10 years, many contributors have developed CAIRO+ [Iskander08, Bourguet04, Tuong06, Dessouky01] under the supervision of Alain Greiner and Marie-Minerve Louërat in LIP6 (Laboratoire d'Informatique Paris 6). CAIRO+ is a language-based design reuse framework that is capable of migrating an IP from one technology to another. The architecture of an IP generator is shown in Fig. 3.9. Each generator possesses four different design sections: The *CREATE* section specifies a netlist template for unsized schematic, a layout template for relative placement and a functional interface consisting of input and output parameters. In the *DESIGN SPACE EXPLORATION* section, the designer documents sizing procedures, explores the design space for

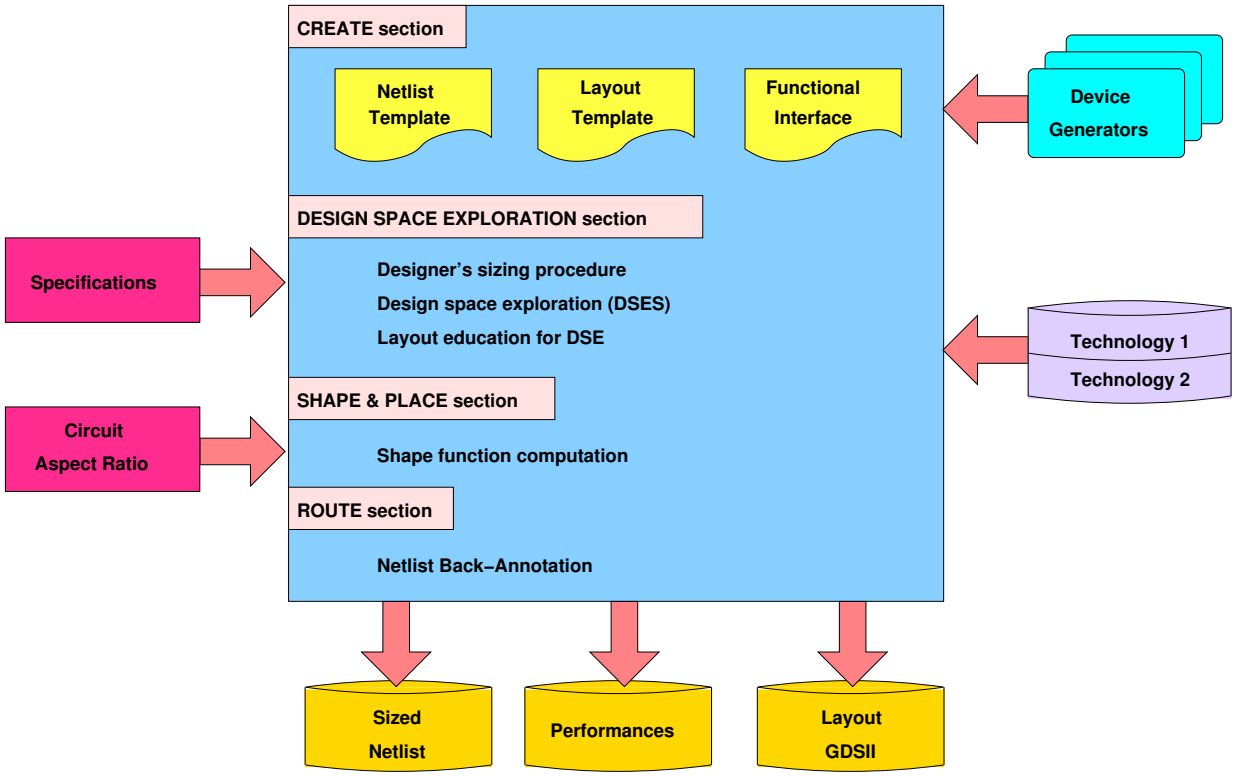

Figure 3.9: *CAIRO+ IP generator architecture.*

trade-offs and evaluates performances in the presence of the generated layout along with parasitic effects. In the *SHAPE & PLACE*, the possible layout configurations are explored to determine the best layout forms that matches geometric constraints on circuit width and height. In the final *ROUTE* section, procedural routing is performed for the generated layout and the netlists back-annotated with the parasitics. During instantiation in a testbench, the IP generator accepts high-level specifications, the required circuit aspect ratio, different technologies and the generators for basic building blocks called *devices*. After execution, the generator generates three different views: a *structural view* in the form of sized netlist, a *behavioral view* in the form of electrical performances and a *physical view* in form of GDSII physical layout.

3.5.6.2 OCEANE: Outils pour la Conception et l'Enseignement des circuits intégrés ANalogiquEs

OCEANE [Porte08] is developed and maintained by Jacky Porte at LIP6 (Laboratoire d'Informatique Paris 6). Since the structure of OCEANE is hierarchical, a design flow should be defined to evolve through this hierarchy. As in the digital domain, the top-down design flow is the ideal design flow for an analog circuit. Starting with a set of high-level specifications, the sizing of elementary components corresponding to the lowest hierarchical level does not require

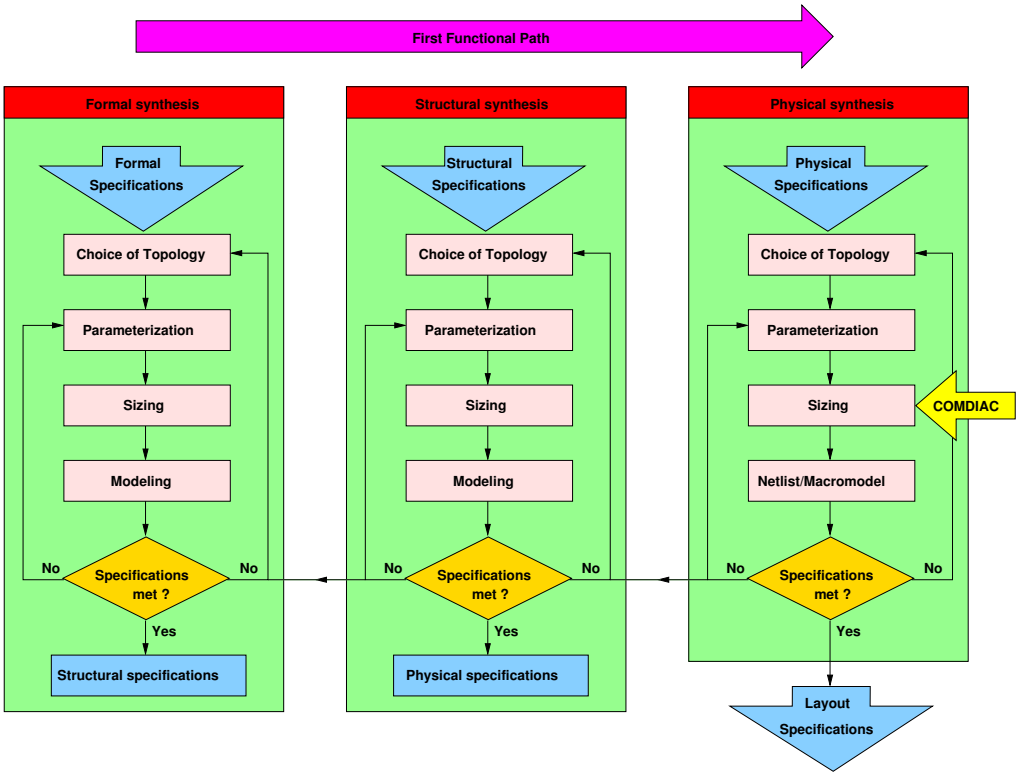

Figure 3.10: *OCEANE design flow [Porte08]* .

a backtrack on a higher-level design phase. The sequence of the different design phases adopted by OCEANE is a *first functional path* followed by step-by-step iterative design phase. Considering the design flow shown in Fig. 3.10, the first functional phase allows to study the feasibility of the realization of the required performances in the highest hierarchical level. Then, the design flow is divided into three phases: *formal synthesis*, *structural synthesis* and *physical synthesis*. For each phase, OCEANE proposes a choice of parameterized topologies, an automated sizing procedure given for one performance specification, an analytical model for the topology and a set of technology models for validating the design using simulation. The technology models range from MOS Level 1 to BSIM3V3 model. If performance specifications are not met, this can be corrected by modifying the topology parameters or by choosing another topology - for either the same or previous hierarchical level.

For the formal synthesis, the parameters for sizing and the simulation models depend on the functional block. For instance, the parameters of a filter may be the coefficients of a transfer function, quality coefficients, ... Modeling is performed using a high-level programming language (typically C language) or using macromodels understood by standard SPICE simulators.

For structural synthesis, the sizing will alter the parameters of an integrator, an amplifier, ..., etc. In filter realization, the parameters will be capacitor ratios and ideal resistances and capaci-

tors, ..., etc. The simulation models are high level models or macromodels.

The physical synthesis phase is independent of the previous synthesis phases. It consists of generating styles for transistors and passive components [Bourguet04].

3.5.6.3 PAD: Procedural Analog Design

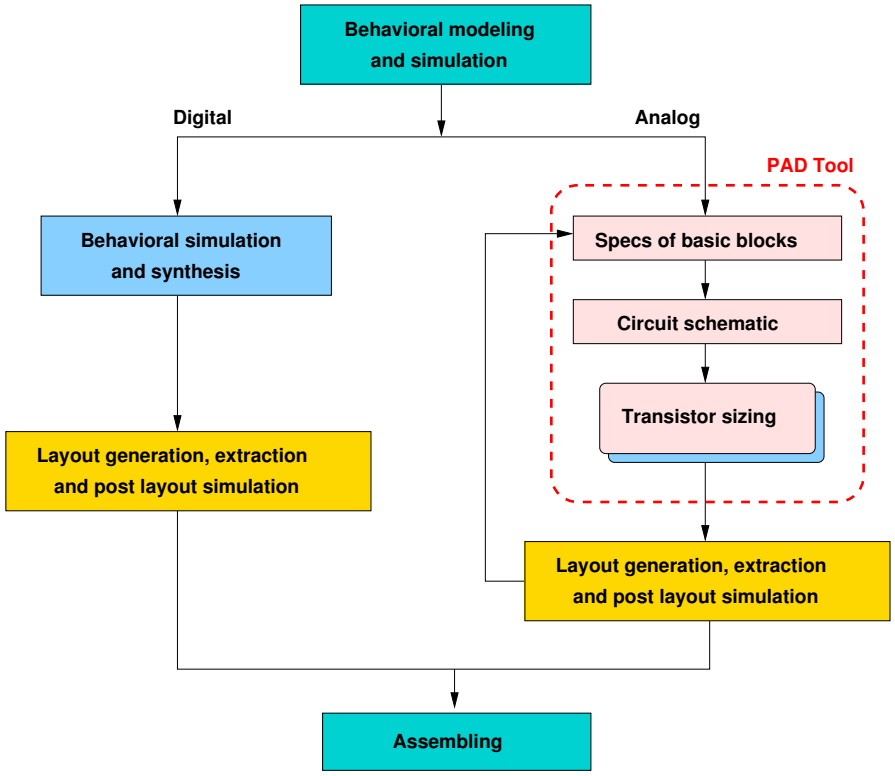

Figure 3.11: *Top-down mixed design approach using PAD [Stefanovic05, Stefanovic03, Stefanovic07, Kayal06].*

PAD [Stefanovic05, Stefanovic03, Stefanovic07, Kayal06] is developed at EPFL (Ecole Polytechnique Federal de Lausanne) under the direction of Prof. Maher Kayal. The introduction of PAD in a mixed design flow is shown in Fig. 3.11. It is a chart-based design environment dedicated to the design of analog circuits aiming to optimize design and quality by finding good trade-offs. This interactive tool allows step-by-step design of analog cells by using guidelines for each analog topology. Its interactive interface enables instantaneous visualization of design trade-offs. At each step, the user modifies interactively one subset of design parameters and observes the effect on other circuit parameters. At the end, an optimized design is ready for simulation (verification and fine-tuning). The present version of PAD covers the design of basic analog structures (one transistor or groups of transistors) and the procedural design of transconductance amplifiers (OTAs) and different operational amplifier topologies. The basic

analog structures calculator embedded in PAD uses the complete set of equations of the EKV MOS model, which links the equations for weak and strong inversion in a continuous way. Furthermore, PAD provides a layout generator for matched substructures such as current mirrors, cascode stages and differential pairs.

3.5.6.4 Seville Design Reuse Flow

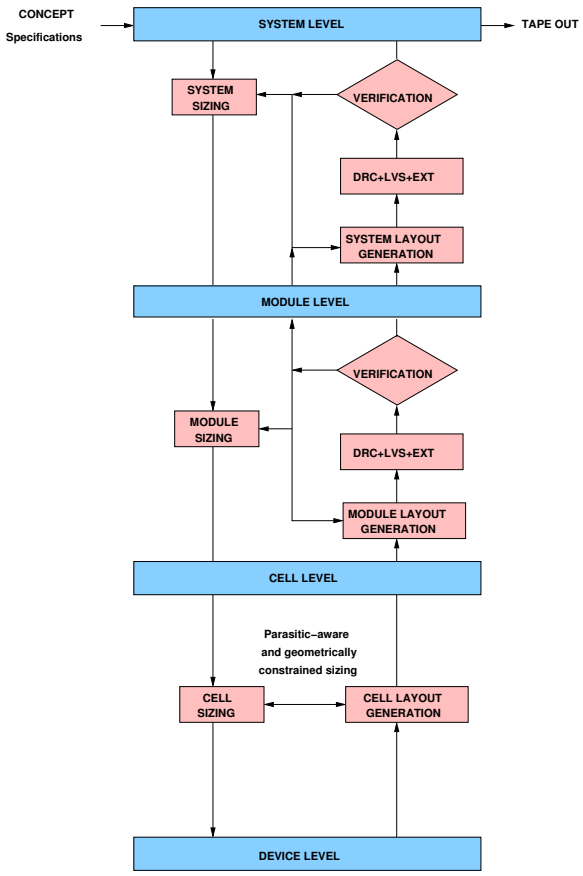

Figure 3.12: *Seville design reuse flow [Lopez04].*

The proposed hierarchical top-down bottom-up reuse flow [Lopez04] shown in fig. 3.12 is developed in university of Seville. It is based on the design flow presented in [Chang97, Gielen00]. During the top-down path, three sizing processes are carried out:

- *System sizing*: where the system-level specifications are transmitted down to obtain performance specifications for every module-level component of the analog mixed-signal system. Because of the complexity of the circuit at this level, behavioral models should be used to speed up the synthesis.
- *Module sizing*: where the performance specifications of each module-level block are mapped

into performance specifications for each of its components. As in system sizing, due to the complexity of the circuit at this level, behavioral models should be used to speed up the synthesis.

- *Cell sizing*: where each cell-level circuit is sized to obtain the value of device-level parameters (transistor width and length, resistor value, capacitor value, ...) such that the performance specifications for the cell-level circuit are properly addressed.

During the bottom-up path and after cell sizing, layout is generated by instantiating the corresponding layout template and the obtained sizing solution, which includes the implementation style of each device if geometrical constraints have been defined. No formal verification is required because the layout template is correct-by-construction. Verification of the extracted layout is neither necessary, since layout parasitics have been already considered. In this way, sizing-layout-sizing spins are avoided, thereby speeding up the overall design reuse flow.

Module layout is then generated by assembling the instantiated layout of its cell-level components or by instantiating the module layout template provided it has been incorporated as a reusable block itself. Only in the former case, formal design rule and layout vs. schematic checks of the module layout are performed. Extraction is necessary in both cases, since no parasitics were considered during module sizing. Afterwards, the module performance is verified by using simulation. Provided that the module-level circuit is not very complex, cell-level components can be replaced by their device-level description. Otherwise, each cell-level circuit can be replaced by a corresponding behavioral model, properly back-annotated with the attained electrical performance (which includes the performance degradation induce by layout). A redesign loop to modify the module layout (i.e. layout elements other than cell layout) or to repeat the module sizing is initiated in the case the module fails to meet the intended performance specifications.

Bottom-up verification of the system-level circuit follows the same methodology. First, the system layout is generated by assembling the module-level components or by instantiating the system-level layout template provided it is available as a reusable block. A formal verification precedes the performance verification where behavioral back-annotated models of the module level circuits (including performance degradation due to parasitics) can be used to reduce the simulation time.

3.5.6.5 Binkley's Transistor Sizing Methodology

The transistor sizing and biasing developed by Binkley [Binkley03] enables optimum design choices throughout the selection of inversion level coefficient *IC* and available channel length *L*. The inversion level coefficient is a normalized measure of MOS drain current that numerically describes the level of channel inversion. The use of the inversion coefficient *IC* allows a knowing and thoughtful selection of MOS operation anywhere in weak, moderate, or strong inversion, which strongly influences all aspects of circuit performance. Operation in the region of low

inversion coefficient *IC* and long channel length *L* results in optimal DC gain and matching compared to the region of high inversion coefficient *IC* and short channel length *L* where bandwidth is optimal. The methodology is implemented in a prototype CAD system where a graphical view permits the designer to explore optimum trade-offs against preset gaols for circuit transconductance g_m, output transconductance g_{ds}, drain-source saturation voltage, gain, bandwidth, white and flicker noise, and DC matching for a $0.5\mu m$ CMOS process. The design methodology can be extended to deeper submicron MOS processes through linkage to the EKV or BSIM3 MOS models or custom model equations.

3.6 Analog IP and Design Representation

In this section, popular design representation for analog intellectual properties are introduced. It will be shown how every representation is used for a different level of design abstraction.

3.6.1 Signal Flow Graphs

Signal flow graphs (SFG) were first introduced by Mason[Mason56]. SFG are extensively used in modeling, simulation, analysis and synthesis of linear electrical networks [Starzyk86, Guindi95, Lee74]. They represent linear performances in the form of linear transfer functions which may be expressed for the circuit level as shown in Fig. 3.13, or at the system (or block level) as shown in Fig. 3.14. Mason also proposed a method known by *Mason's gain formula* [Mason56] that computes the overall transfer function from the SFG graphical representation as shown in the last step in Fig. 3.13. Contemporary symbolic analysis techniques [Gielen89, Gielen94, Gielen91] use SFGs to symbolically express the behavior of very complex analog circuits.

3.6.2 Bipartite Graphs

The bipartite graph was used by Gielen *et al*[Gielen93, Swings91c, Plas01] to represent the expressions *direct current (DC), alternating current (AC), transient, etc.* that fully describe the relationships between the circuit behavior and the circuit parameters. These equations are declarative, i.e., they only specify relationships that must hold simultaneously between different variables, but they do not describe a direction nor sequence of solution (they are not assignments). DC equations are derived automatically from the circuit topology; AC equations are derived by means of symbolic analysis techniques like with the ISAAC [Gielen89] or SYMBA [Wambacq95] tools; transient and other equations to date still have to be provided by the designer. In this way, most of a declarative model can be generated automatically. The resulting model however is still declarative and, therefore, not yet suited for computer execution. The equation manipulation tool DONALD [Gielen93, Swings91c, Plas01] is, therefore, used to automatically determine the degrees of freedom in the design, then to choose a set of independent input variables (equal to the number of

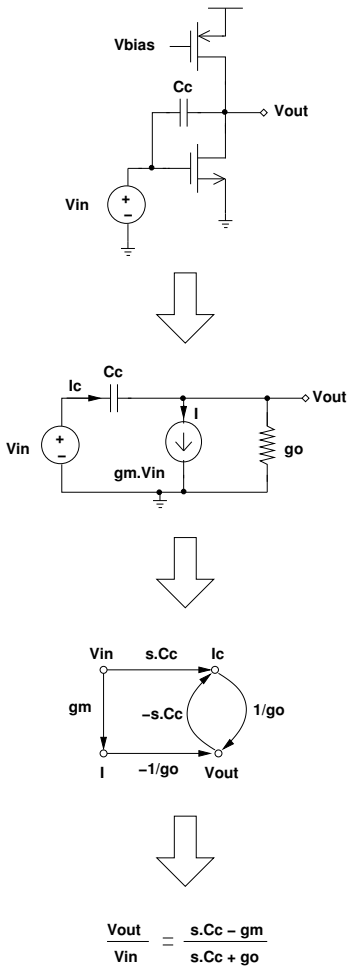

Figure 3.13: *Signal flow graph for a MOS amplifier.*

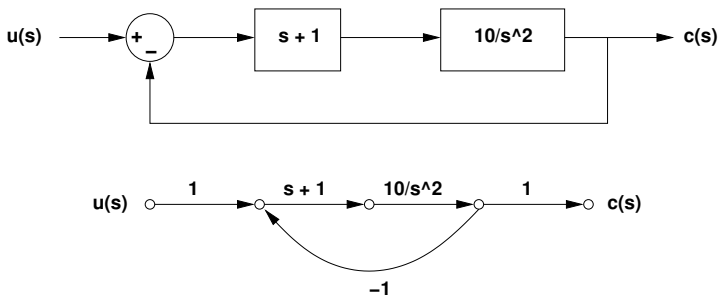

Figure 3.14: *Signal flow graph for a two-block system with negative feedback.*

degrees of freedom), and then to turn the undirected declarative model into a directed sequential computation plan, which indicates how (by means of which equations, in which direction, and in which sequence) all the dependent variables have to be calculated from the values of the

independent ones.

Example : The approach is now illustrated for a simplified example to demonstrate the concepts. Fig. 3.16 shows the DC part in equations (3.4)-(3.13). This corresponds to the declarative model the common source single-transistor amplifier with resistive load shown in Fig. 3.15. The equations use the variable notation <*quantity*>.<*terminal*>.<*instance*>. The declarative model is represented as a bipartite graph containing two different types of vertices: ovals for the variables, rectangles for the constraining equations. Note that the graph is undirected. The declarative equations are shown in the equation at the bottom of the next page. In total, there are 14 variables constrained by 10 independent equations. Hence, there are four degrees of freedom in this simplified example. This means that four independent variables can be selected as input variables. Many combinations are possible. If we choose, for instance, the variables $v.vdd, \log(l.mn), \log(w.mn), vgs.mn$ as input set, then the originally undirected bipartite graph can be directed using constraint propagation techniques, as shown in Fig. 3.17, indicating the direction and order in which the equations have to be solved in order to calculate the values of the remaining ten dependent variables (single line ovals) out of the values of the independent variables (double line ovals).

$$
\begin{aligned}
eq\_vgnd &: v.gnd = 0 && (3.4)\\
eq\_vout &: vout = \frac{v.vdd - v.gnd}{2} && (3.5)\\
eq\_vsb.mn &: vsb.mn = v.s.mn - v.b.mn && (3.6)\\
eq\_vds.mn &: vds.mn = v.d.mn - v.s.mn && (3.7)\\
eq\_ids.mn &: ids.mn = f(\log(l.mn), \log(w.mn), vgs.mn, vds.mn, vsb.mn) && (3.8)\\
eq\_i.rl &: i.rl = \frac{v.p.rl - v.n.rl}{rl} && (3.9)\\
eq\_kcl.out &: i.d.mn + i.n.rl = 0 && (3.10)\\
eq\_i.d.mn &: i.d.mn - ids.mn && (3.11)\\
eq\_i.s.mn &: i.s.mn = -ids.mn && (3.12)\\
eq\_kcl.gnd &: i.s.mn + i.gnd = 0 && (3.13)\\
& && (3.14)
\end{aligned}
$$

3.6.3 Platform-Based Design

Platform-based design (PBD) as presented in [Rabaey06, Keutzer00, Balarin03] has emerged as a novel paradigm in the digital domain to allow designing at higher level of abstraction while considering lower level physical properties. The PBD paradigm is a meet-in-the-middle approach consisting of a bottom-up characterization phase and a top-down mapping phase. A platform is a

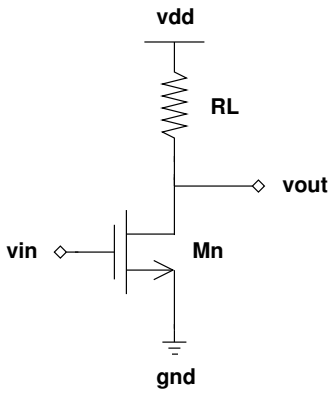

Figure 3.15: *Example circuit to illustrate design plan computation.*

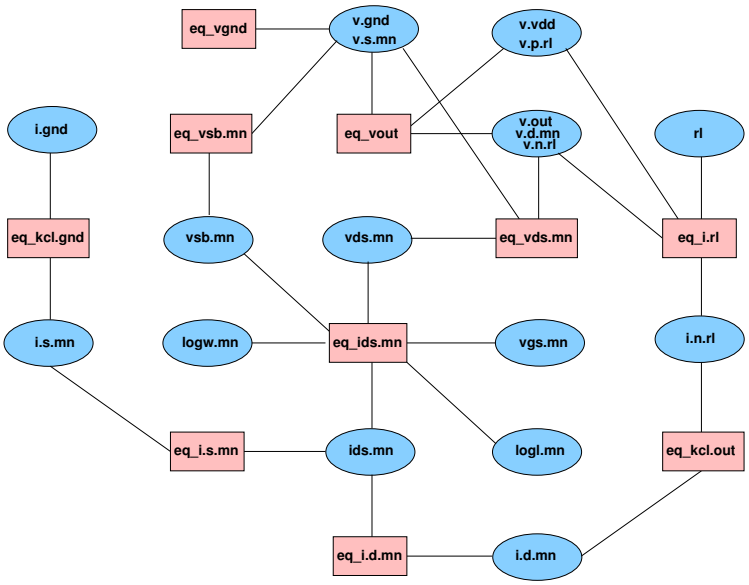

Figure 3.16: *Undirected bipartite graph of circuit in Fig. 3.15.*

library of components and interconnects along with composition rules, determining legal compositions of components (platform instances). The bottom-up characterization phase abstracts architectures as library components providing a set of models for the services that can be implemented on it and their cost and performance. The top-down phase consists of selecting the optimal platform instance (according to some cost function) that can support the requested functionality while satisfying all system and architecture constraints.

In this paradigm, an analog platform (AP) is a set of components each decorated with a set of behavioral models (**F**), configuration and performance models (**C**, **P**), and validity laws (**L**) [Bernardinis05a]. This rich set of models helps to address the concerns raised earlier, leveraging behavioral models to perform design optimization while considering architectural constraints and

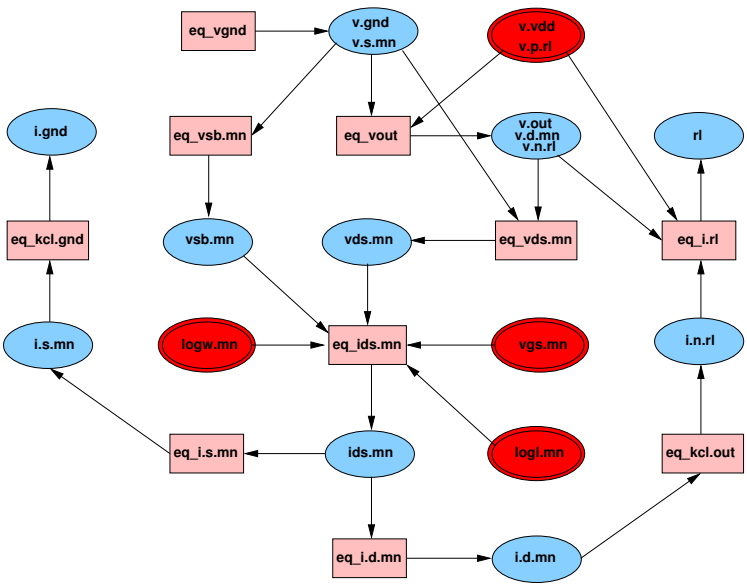

Figure 3.17: *Directed bipartite graph of circuit in Fig. 3.15.*

costs. Essential also is that the resulting abstractions ensure the following properties:

- *Flexibility*: any analog component can be encapsulated as an AP: newly specified circuits, analog IPs (possibly third party s), module generators, circuit synthesizers and optimizers.
- *Accuracy*: the AP abstraction requires a set of models that introduce architectural effects at the system level while guaranteeing composability.
- *Hierarchy*: AP components allow building high level hierarchical models while preserving information on the actual architecture space. Therefore, correct abstraction levels can be selected for MS designs and enable efficient design space explorations.
- *Implementability*: a notion of feasible performance is propagated bottom-up into the design hierarchy, thus restricting (and characterizing) the actual design space.

The *behavioral model* **F** allows for the abstract computation of the system response without being directly constrained to a specific architecture. Very general techniques can be adopted to implement behavioral models, ranging from hand written block models (requiring deep insight on the designer/developer) to model order reduction approaches, which are based on a mathematical formulations and can be fully automated [Roychowdury04].

Even more essential to the AP abstraction is the introduction of *configuration* (**C**) and performance (**P**) models. For a given module or component, the configuration model outlines the space of the feasible realizations (in terms of design parameters such as transistor sizes, bias currents, supply voltages, etc.). The corresponding performance model maps these configuration

constraints into a set of feasible performance vectors. For example, for OTAs, the performance model is specified in terms of the *gain, noise, bandwidth, power* n-tuples, which accurately identify the feasible performance range of a given OTA architectures in a given technology. Having quantified bounds available is more attractive and reliable than the recursive estimation and optimization based approaches (such as advocated in [Roychowdury04]), especially in light of the many secondary parameters emerging in today's deep sub-micrometer processes. In hierarchical designs, where several components are connected together (defining a platform instance), the performance model of level l is the configuration space of level $l+1$, hence a direct relation exists between the performance model $\mathbf{P}^l$ and the configuration model $\mathbf{C}^{l+1}$.

The efficient and accurate mapping of configuration models into performance models is one of the main challenges of the A-PBD approach. Traditional performance models are based on regression schemes, for which a rich literature exists (ranging from simple quadratic models for optimization [Brayton81] to advanced data mining techniques [Liu02] and template independent schemes [McConaghy05]). A more effective strategy is based on *classifiers*, which have the distinct advantage of making it possible to encapsulate architectural alternatives for the same functionality (that obviously share the same performance space) in a very straightforward manner. Only a negligible setup time is required to define a performance model for arbitrary performance figures and circuit topologies can be used. On the other hand, since the approach is based on a sampling scheme requiring accurate simulation of performance, the characterization itself may be expensive. In [Bernardinis05b], an approach that uses bipartite graphs called *analog constraint graphs* (ACG) (similar to the previous section but augmented with inequality constraints), aimed at pruning the number of samples (simulations) required to characterize a circuit is presented exploiting structural and functional properties of the configuration space. Even if the exponential nature of the problem is not affected, the approach has shown to be practical for real case studies. In addition, the process is easily parallelizable, so the entire characterization time can be reduced to a few machine hours. A classification approach for analog performance based on support vector machines [Boser92] is presented in [Bernardinis03].

Validity laws $\mathbf{L}$ form the final element of the AP abstraction. When assembling a platform instance (composing platform library elements), the accuracy of the instance model has to be guaranteed. In fact, the plain composition of behavioral models may not correspond to the behavioral model of the composition. Validity laws limit the scope of behavioral models to enforce correct compositions and accurate modeling of interface effects (e.g., circuit loading due to other circuits). An example of validity laws and interface modeling in the AP context can be found in [Bernardinis04], where an RF receiver platform is built and used to optimize a UMTS system.

The essence of platform-based design is building a set of abstractions that facilitate the design of complex systems by a successive refinement/abstraction process [Rabaey06, Bernardinis05a] as shown in Fig. 3.18. The top left graph shows the analog constraint graph (ACG) used to sample performance of the telescopic operational transconductance amplifier (OTA)

3.7 Conclusion

We conclude that analog design automation is still a very hot and not fully covered topic. Many standardization and research efforts are still needed to achieve an acceptable level of maturity. One potential research topic is the development of an efficient analog design representation. This could strongly provide lots of insight for analog design and reuse tools. In this thesis work, we investigate the problem of design representation and its impact on future EDA tools. We propose a design representation through which many aspects of analog design can be identified and analyzed.

Table 3.2: *Comparison of Synthesis Tools [Ochotta98].*

	Synthesis Tool							
	OPASYN [Koh87]	**OASYS [Harjani87]**	**IDAC [DeGrauwe84b]**	**ARIADNE [Swings91a]**	**STAIC [Harvey92]**	**ISAID [Makris92]**	**Maulik [Maulik91]**	**AMGIE [Plas01]**
School	University of California at Berkeley	Carnegie Mellon University	Centre Suisse d'Electronique et de Microelectronique	Katholieke Universiteit at Leuven	University of Waterloo	Imperial College in London	Carnegie Mellon University	Katholieke Universiteit at Leuven
Performance evaluation	Equations	Equations	Equations	Equations	Equations	Equations/ Qualitative Reasoning	Equations	Equations
Non-Linear device models	Simplified equations	Simplified equations	Custom models	Simplified equations	Simplified equations	Simplified equations	Equations+BSIM	Equations+BSIM
Worst-case accuracy	200%	25%	15%	10%(low-perf.)	24%	14%	24%	Not reported
Search methods	Gridded, Steepest descent optim.	Plan steps with backtracking	Plan steps with post optim.	Simulated annealing	Coarse initial optim. + detailed final optim.	Qualitative reasoning + post optim.	Sequential quadratic programming (SQP)	VFSR, Hooke-Jeeves, minimax, or SQP
Synthesis time (approximate)	1 min	5 sec	A Few seconds	5 min	3 min	Not reported	1 min	few minutes to 2 hour
Machine	VAX 8800	VAX 8800	CPU N/A	CPU N/A	MIPS 2000	Not reported	DEC 3100	SUN Ultra 1-170, HP 712/80 UNIX
Preparatory effort to add new circuit	2 weeks for well understood circuit	6 months including circuit analysis	4-45 designer-months	Not reported	3 circuits required 100000 lines of code	Not reported	6 months including circuit analysis	1 week
Equations derived	Manually	Manually	Symbolic simulation + Manually	Symbolic simulation + Manually	Manually	Manually	Manually	Manually + Symbolic analysis
How/Where Equations are stored	Hard-coded	Hard-coded	Hard-coded + Design files	Not reported	language + Database	Not reported	Hard-coded	Hard-coded
Most Complex circuit example	7 variables (60 device opamp)	19 variables (17 device opamp)	N/A (15 circuit types, incl. delta-sigma, from 5-30 devices)	14 variables (9 device opamp)	N/A (22 device opamp)	8 variables (13 device opamp)	39 variables, 7 for topology selection (19 devices opamp)	N/A (Particle Detector Front-End)

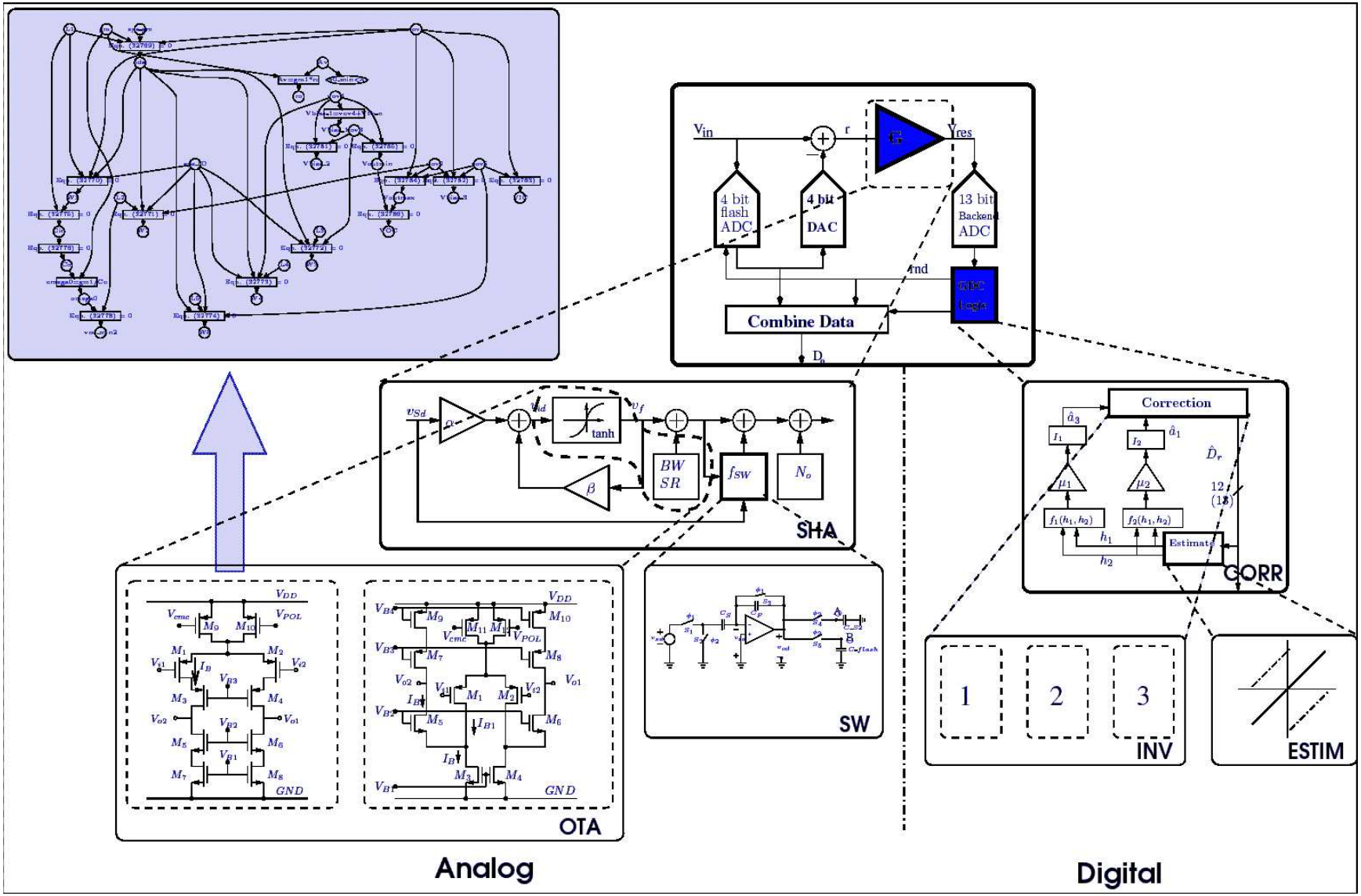

Figure 3.18: *Mixed-signal platform based design: Starting from the bottom left corner, an analog platform stack is built from circuit level components generating instances and new components at higher levels of abstraction. The top left graph shows the analog constraint graph (ACG) used to sample performance of the telescopic operational transconductance amplifier (OTA). The digital part of the mixed signal platform is generate in a similar way as shown on the right. .*

Chapter 4

Transistor Sizing and Biasing Methodology

4.1 Introduction

In this chapter, a complete formulation for the transistor sizing and biasing that unifies both standard simulation method and operating point driven formulation, will be presented. In the proposed formulation, a library of procedures for computing the sizes and biases of a transistor are developed. These procedures are essentially based on accurate standard BSIM3V3 models. They serve as the interface between our proposed reuse strategy and the technology. The procedures are overloaded to implement both standard simulation method and operating point driven formulation. Incorporating these procedures into any knowledge-based synthesis system, allows it to solve for transistor sizes and biases very accurately. The procedures were implemented in our dedicated framework CAIRO+ [Dessouky01, Tuong06]. The ideas behind the procedures are partially inspired from the work done for standard simulation methods [Vlach94] and operating point driven formulation [Plas01, Leyn98].

In section 4.2, a brief description is given about the task of the BSIM3V3 model integration performed in CAIRO+.

In section 4.3, since our transistor sizing and biasing methodology relies on inverting the BSIM3V3 model, we present the concept of *sizing and biasing operators* performing the BSIM3V3 model inversion.

In section 4.4, the BSIM3V3 MOS engine of CAIRO+ is enhanced y incorporating sizing and biasing operators.

In section 4.5, we illustrate an example for transistor sizing and biasing.

Finally, section 4.6 concludes our proposed methodology.

4.2 BSIM3V3 Model Integration

In order to maintain a good precision that is comparable to a simulator, it becomes mandatory to integrate a high-precision BSIM3V3 model into CAIRO+. This ensures that our framework controls the design errors that appears in early design phases. The initial integration of the BSIM3V3 in CAIRO+ was done by Laurent de Lamarre [de Lamarre02]. The model was then enhanced as follows:

1. To allow the code to be easily explored by future developers, the OCEANE model structure was preferred. The OCEANE model structure presented a better decomposition of functions that facilitates the debugging of the model.

2. The binning was done only on the parameters needed, hence, a huge time saving during simulation and optimization is gained.

3. Some small-signal capacitances were computed numerically in the initial model. This has been replaced by complete analytical equations that were derived from the BSIM3V3 equations and tested against the results of a commercial simulator.

To allow technology migration by switching to different technology files, a complete characterization and debugging of the model over different technologies (namely 0.35μ 0.6μ and 0.13μ) was then conducted. In addition, the technology parameter files and configuration files for the 0.13μ ST CMOS technology have been converted from SPICE file formats to C files and were compiled and linked to the MOS engine.

It is clear that the task of integrating new technology models into the CAIRO+ framework is a very laborious and error-prone task. Since the compact device model is expected to be accurate, it requires lot of iterations for characterization and debugging.

4.3 Sizing and Biasing Operators

In this section, we will describe the concept of sizing and biasing operators. We start with the principal idea, then we describe how to define and implement one operator. Finally, the list of implemented operators will be given.

4.3.1 Principal Idea

Assume that the width W of an NMOS transistor is to be computed at ambient temperature $Temp$. In strong inversion, the quadratic model of the MOS transistor is:

$$I_{DS,n} = \frac{\mu_n}{2} C_{ox} \frac{W}{L} (V_{GS,n} - V_{th,n})^2 (1 + \lambda_n V_{DS,n}) \quad (4.1)$$

$$= \frac{\mu_n}{2} C_{ox} \frac{W}{L} (V_{eg,n})^2 (1 + \lambda_n V_{DS,n}) \quad (4.2)$$

$$V_{th,n} = V_{th0,n} + \gamma_n (\sqrt{2\phi_F - V_{BS,n}} - \sqrt{2\phi_F}) \quad (4.3)$$

$$\phi_F = \frac{k \cdot Temp}{q} \ln \frac{N_A}{n_i} \quad (4.4)$$

where the μ_n is the mobility of electrons near the silicon surface, C_{ox} is the gate oxide capacitance per unit area, λ_n is the channel-length modulation coefficient, $V_{th0,n}$ is the threshold voltage with zero V_{BS}, γ_n is the body-effect constant, N_A is the substrate doping concentration and n_i is the intrinsic carrier density. This very simple model is based on device physics appropriate for long-channel and uniform doping. Because the model equations are simple and easy to understand, they are still used for hand calculations and preliminary circuit simulations. However, the model has poor accuracy and scalability [Cheng99].

To compute W from equation (4.1), the following design parameters should be set: $I_{DS,n}$, L, $V_{GS,n}$, $V_{th,n}$ and $V_{DS,n}$. Normally, $I_{DS,n}$ and L are set by the designer. Since $V_{eg,n} = V_{GS,n} - V_{th,n}$, setting $V_{eg,n}$ fixes the transistor region of operation. Designers may choose to set $V_{eg,n}$ in equation (4.2), instead of $V_{GS,n}$ and $V_{th,n}$ in equation (4.1) as in [Silveira96, Porte08]. Equation (4.3) shows that $V_{th,n}$ is controlled by $V_{BS,n}$ which should be also fixed by the designer. Notice from equation (4.4) the dependence of the surface inversion potential $2\phi_F$, hence $V_{th,n}$, on the temperature $Temp$. To summarize, $Temp$, I_{DS}, L, V_{eg} or V_{GS} and V_{BS} are set by the designer.

The only remaining parameter that needs to be set is V_{DS}. This parameter is either set as part of transistor specifications or fixed through the topology connections.

Based on the above considerations, an arbitrary analog circuit can be viewed as a set of connections fixing the drain potentials of all transistors. Actually, few connections are possible to fix the drain potential. Let us assume that an arbitrary circuit consists of transistors M_1, M_2 and M_3 as in Fig. 4.1 . To size the three transistors, the potential of the common drain node should be fixed. The most common connections to fix this potential are:

1. The drain is considered as an input terminal. Its DC value is fixed by the designer.
2. The drain is considered as an output terminal. Its DC value is fixed by the designer.
3. The drain is connected to the source of transistor M_A. Its DC value is set to V_{S,M_A} which becomes the unknown.
4. The drain is connected to the gate of transistor M_B. Its DC value is set to V_{G,M_B} which becomes the unknown.

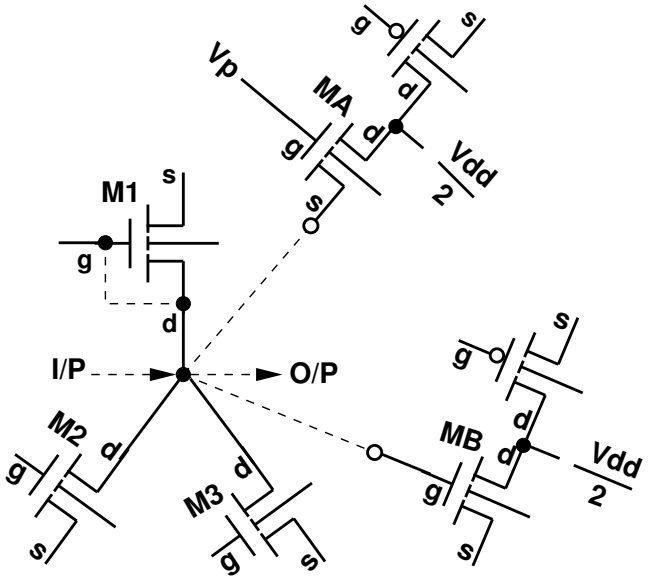

Figure 4.1: *Possible drain connectivity*.

5. The drain is connected to a gate, forming a diode-connected transistor as M_1. The gate/drain voltage $V_{G/D}$ becomes the unknown.

Points (3), (4) and (5) are solved by inverting the standard transistor model. Instead of using the quadratic model, the standard BSIM3v3 transistor model has been integrated into our framework . In other words, the BSIM3v3 analytical model is numerically inverted in order to solve for V_S, V_G and $V_{G/D}$ in terms of V_D. The study of the inversion of the BSIM3v3 transistor model resulted in developing the sizing and biasing operators described in subsequent sections.

4.3.2 Operator Definition

As discussed in the previous subsection, it is essential to our methodology to solve for the unknown electrical parameters V_S, V_G, $V_{G/D}$, as well as I_{DS}, W and V_{th}. I_{DS} and V_{th} are provided explicitly by the BSIM3v3 model equations. V_S, V_G, $V_{G/D}$ and W, are to be solved for by numerically inverting the BSIM3v3 model equations. Inversion operations are performed by a set of operators called *sizing and biasing operators*. The study of model inversion resulted in implementing 46 sizing and biasing operators in CAIRO+ framework. Each sizing and biasing operator has the following form:

$$OP{<}class{>}(RV_i, ...) : (LV_j, ...) \Leftarrow (RV_n, ...) \tag{4.5}$$

where $<class>$ is the main electrical parameter to be computed, RV_i is a subset of the known electrical parameters that dictates which operator version to apply, RV_n is the set of all known electrical parameters required by the operator to execute, and LV_j is the set of unknown electrical

parameters that are computed by this operator. It is said that a parameter is known, if it is either fixed by the designer or previously computed by CAIRO+ during sizing.

Table 4.1: *Class definition of sizing & biasing operators*.

Operator	Definition
OPVS(V_{eg}, V_B)	$(V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G, V_B$
OPVS(V_{eg})	$(V_S, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G$
...	...
OPVG(V_{eg})	$(V_G, V_B, W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S$
...	...
OPVGD(V_{eg})	$(V_G, V_D, V_B, W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S$
...	...
OPW(V_G, V_S)	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_D, V_G, V_S$
...	...
OPIDS(V_{eg})	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_{eg}, V_D, V_S$
...	...

Table 4.1 shows the definition of the main five classes of the sizing and biasing operators applied to the reference transistor. Let us examine in further detail one operator such as $OPVS$. The $OPVS$ operator class is *source voltage*. The table shows only two versions of this operator. The first version $OPVS(V_{eg}, V_B)$ is called whenever V_{eg} is known and the reference transistor is not bulk-source connected i.e. V_B should be fixed by the designer. This operator computes V_S, V_{th} and W, simultaneously, in terms of $Temp$, I_{DS}, L, V_{eg}, V_D, V_G and V_B. The second version $OPVS(V_{eg})$ is called whenever V_{eg} is known and the reference transistor is bulk-source connected. This operator also determines V_S, V_B, V_{th} and W, simultaneously in terms of $Temp$, I_{DS}, L, V_{eg}, V_D and V_G.

4.3.3 BSIM3V3 Model Inversion

For the BSIM3V3 model, the characterization curve of I_{DS} versus V_{GS} has been simulated for different V_{DS}, as shown in Fig. 4.2. It is clear from the curve that I_{DS} is a monotonic function of V_{GS}.

Consequently, solving for V_{GS} can be easily done using the following Newton-Raphson equation:

$$v_{gs}^{k+1} = v_{gs}^{k} - \frac{I_{ds}(v_{gs}^{k}) - I_{ds,desired}}{g_m(v_{gs}^{k})} \tag{4.6}$$

In our actual implementation for this problem, we adopted a variant of the *Damped Newton-Raphson* strategy [Coughran83]. We chose to use the direction corruptive damping scheme for

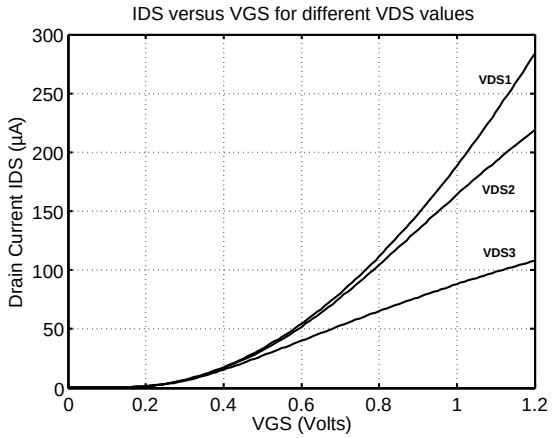

Figure 4.2: *Plot of I_{DS} versus V_{GS} for different $V_{DS1} > V_{DS2} > V_{DS3}$.*

solving for the voltage V_{GS}:

$$v_{gs}^{k+1} = v_{gs}^{k} + \min(\gamma, \mid \frac{I_{ds}(v_{gs}^{k}) - I_{ds,desired}}{g_m(v_{gs}^{k})} \mid) \times sign(-\frac{I_{ds}(v_{gs}^{k}) - I_{ds,desired}}{g_m(v_{gs}^{k})}) \tag{4.7}$$

where γ is an arbitrary constant $0 < \gamma < 1$. In order to evaluate $I_{ds}(v_{gs}^{k})$, the BSIM3V3 model equation *BSIM3V3_IDS(...)* is called. We also note that the first derivative of I_{DS} with respect to V_{GS} is simply the transconductance g_m of the MOS transistor. The transconductance is evaluated by calling the actual BSIM3V3 model function *BSIM3V3_GM(...)*. This is done to evaluate the derivative analytically rather than numerically.

The same observation is made for the characterization curve of I_{DS} versus W, shown if Fig. 4.3. It is clear that I_{DS} is monotonic function of W.

In this case, the traditional newton Raphson algorithm is used to solve for the width W:

$$w^{k+1} = w^{k} - \frac{I_{ds}(w^{k}) - I_{ds,desired}}{I'_{ds}(w^{k})} \tag{4.8}$$

Based on these observations, each sizing and biasing operator is designed to solve only one of the two numerical problems discussed above. Since these two numerical problems are guaranteed to always converge due to the monotonicity of I_{DS} with respect to V_{GS} and W, the sizing and biasing operators do not suffer from convergence problems existing in analog simulators. Rather, if the design parameters given to the operators are not consistent, the operator can diagnose the direct cause of inconsistency and prints an diagnostic message to the designer.

4.3.4 Convergence Criteria

In order to stop the Newton-Raphson iterations, the same convergence criteria that are integrated into spice simulators were adopted. The convergence criteria for voltage, current, width and

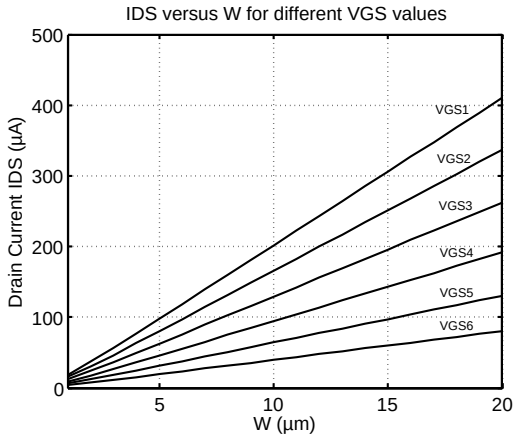

Figure 4.3: *Plot of I_{DS} versus W for different $V_{GS1} > V_{GS2} > V_{GS3} > V_{GS4} > V_{GS5} > V_{GS6}$.*

length quantities, are respectively:

$$| v_n^{(k)} - v_n^{(k-1)} | < reltol \cdot v_{nmax} + vntol \quad (4.9)$$
$$where \;\; v_{nmax} = max(| v_n^{(k)} |, | v_n^{(k-1)} |)$$

$$| i_n^{(k)} - i_n^{(k-1)} | < reltol \cdot i_{nmax} + abstol \quad (4.10)$$
$$where \;\; i_{nmax} = max(| i_n^{(k)} |, | i_n^{(k-1)} |)$$

$$| w_n^{(k)} - w_n^{(k-1)} | < reltol \cdot w_{nmax} + abstol \quad (4.11)$$
$$where \;\; w_{nmax} = max(| w_n^{(k)} |, | w_n^{(k-1)} |)$$

$$| l_n^{(k)} - l_n^{(k-1)} | < reltol \cdot l_{nmax} + abstol \quad (4.12)$$
$$where \;\; l_{nmax} = max(| l_n^{(k)} |, | l_n^{(k-1)} |)$$

Defaults values for $reltol$, $vntol$ and $abstol$ are 1.0e-5, 1.0e-6 and 1.0e-12 respectively.

4.3.5 Operator Implementation

To illustrate how to build an operator, the implementation of the operator *OPVS(V_{eg})* is given in Fig. 4.4:

- In line 1, the operator name is declared.
- In line 2, the input parameters required by the operator are specified.

1 **Operator** OPVG(V_{eg})
2 **Inputs** $Temp$,V_S,V_D,V_{eg},L,I_{DS}
3 **Outputs** V_G,V_B,W,V_{th}
4 **Implements**
5 **Initialize** W_{min} from technology file
6 **Let** $V_B = V_S$
7 **Let** $V_{DS} = V_D - V_S$
8 **Let** $V_{BS} = V_B - V_S = 0.0$
9 **Let** $V_{th} = BSIM3V3\_VTH(Temp, L, n \cdot W_{min}, 0.0, V_{DS}, V_{BS})$
10 **Let** $V_G = 0.0$
11 **Let** $W = W_{min}$
12 **Let** iteration_count = 0
13 **Do**
14 **Set** $W_{prev} = W$
15 **Set** $V_{G,prev} = V_G$
16 **Set** $V_G = V_S + V_{eg} + V_{th}$
17 **Set** $V_{GS} = V_G - V_S$
18 **Solve** for W using equation (4.8) for $I_{DS,desired} = I_{DS}$
19 **Set** $V_{th} = BSIM3V3\_VTH(Temp, L, W, V_{GS}, V_{DS}, V_{BS})$
20 **Increment** iteration_count
21 **While** $\mid V_G - V_{G,prev} \mid \geq \epsilon_v$ **and** $\mid W - W_{prev} \mid \geq \epsilon_w$ **and** Maximum Iterations count not reached.
22 **Return** $[V_G, V_B, W, V_{th}]$

Figure 4.4: *Implementation of the operator OPVG(V_{eg}).*

- In line 3, the output parameters computed by the operator are specified.
- In line 4, the body of the operator is defined.
- In line 5, the minimum width of the technology W_{min} is initialized from technology files.
- In line 6, since this operator assumes that the bulk and source are connected, then they have equal node potential.
- In line 7, the differential drain-source voltage is initialized.
- In line 8, the differential bulk-source voltage is initialized to zero since the bulk and source are assumed connected.
- In line 9, an estimate of the threshold voltage V_{th} is computed for an arbitrary width $n \cdot W_{min}$ and $V_{GS} = 0$.
- In line 10, the gate voltage is initialized to zero,

- In line 11, the width is initialized to the minimum width.
- In line 12, iteration count is initialized to zero.
- In line 13, the loop computing the unknowns V_G, V_B, W and V_{th} starts here.
- In line 14, the previous computed width is preserved.
- In line 15, the previous computed gate voltage is preserved.
- In line 16, a new gate voltage is computed from the overdrive voltage V_{eg} and the threshold voltage V_{th}. The computation follows the $\frac{g_m}{I_D}$ principle presented in [Silveira96].
- In line 17, the difference gate-source voltage is computed.
- In line 18, the width is solved for using equation (4.8) for the desired current I_{DS}.
- In line 19, the BSIM3V3 model function *BSIM3V3_VTH* is called to compute the actual V_{th}.
- In line 20, the iteration count is increased by 1.
- In line 21, the convergence is tested using equations 4.9 and 4.11. ϵ_v and ϵ_w represent the right-hand side of equations 4.9 and 4.11, respectively. The maximum iterations count is tested to stop the loop if a maximum count is reached.
- In line 22, the computed unknowns $[V_G, V_B, W, V_{th}]$ are returned back to the caller.

Note that in line 19, the actual BSIM3V3 model was called in order to take into account deep submicron effects affecting the threshold voltage discussed in appendix A. The operator was designed to compute the threshold voltage that occurs from the designer required sizing and biasing conditions. The accurate threshold voltage is always considered as an output from the sizing and biasing operators. This way design errors due to threshold voltage estimation can be eliminated during the early sizing and biasing phases.

4.3.6 Library of Operators

As discussed in subsection 4.3.1, it is required to numerically invert the BSIM3v3 analytical model in order to solve for V_S, V_G and $V_{G/D}$. This is done by implementing additional operators, based on the principles discussed in previous sections. A complete library consisting of 46 sizing and biasing operators was developed and integrated into the CAIRO+ framework. For example, Table 4.2 lists the operators used to compute the source node voltage of a MOS transistor. The table is divided into two groups. The first group is used in the case of the bulk-source connected transistor. The second one is used in the case of bulk and source are not connected. In each group, one can notice that some operators do not take width W as input. These operators account for the

Table 4.2: *Definition of Operators for V_S Computation.*

Operator	Definition
OPVS(V_{eg})	$(V_S, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G$
OPVS(V_{GS})	$(V_S, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_G$
OPVS(V_{eg}, W)	$(V_S, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_D, V_G$
OPVS(V_{GS}, W)	$(V_S, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_D, V_G$
OPVS(V_G, W)	$(V_S, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_D, V_G$
OPVS(V_{eg}, V_B)	$(V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G, V_B$
OPVS(V_{GS}, V_B)	$(V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_G, V_B$
OPVS(V_{eg}, V_B, W)	$(V_S, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_D, V_G, V_B$
OPVS(V_{GS}, V_B, W)	$(V_S, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_D, V_G, V_B$
OPVS(V_G, V_B, W)	$(V_S, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_D, V_G, V_B$

Table 4.3: *Definition of Operators for V_G Computation.*

Operator	Definition
OPVG(V_{eg})	$(V_G, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S$
OPVG(V_{GS})	$(V_G, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_S$
OPVG(V_{eg}, W)	$(V_G, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_D, V_S$
OPVG(V_{GS}, W)	$(V_G, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_D, V_S$
OPVG(V_S, W)	$(V_G, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_D, V_S$
OPVG(V_{eg}, V_B)	$(V_G, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S, V_B$
OPVG(V_{GS}, V_B)	$(V_G, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_S, V_B$
OPVG(V_{eg}, V_B, W)	$(V_G, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_D, V_S, V_B$
OPVG(V_{GS}, V_B, W)	$(V_G, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_D, V_S, V_B$
OPVG(V_S, V_B, W)	$(V_G, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_D, V_S, V_B$

operating-point-driven formulation. The operators that accepts the width as input, account for the standard simulation approach.

Similarly, Table 4.3 and Table 4.4 list the operators used to compute V_G and $V_{G/D}$ respectively.

Another set of operators that compute the width is shown in Table 4.5. These operators are used if the only unknown to be computed is the width. This is essentially used for the operating point driven formulation when the only unknown to be computed is the width of a transistor.

The latest set of operators that computes current of MOS transistor is listed in Table 4.6. This is used for both operating point formulation and standard simulation approach when the only unknown that needs to be computed is the current.

Table 4.4: *Definition of Operators for $V_{G/D}$ Computation.*

Operator	Definition
OPVGD(V_{eg})	$(V_G, V_D, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S$
OPVGD(V_{GS})	$(V_G, V_D, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_S$
OPVGD(V_{eg}, W)	$(V_G, V_D, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_S$
OPVGD(V_{GS}, W)	$(V_G, V_D, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_S$
OPVGD(V_S, W)	$(V_G, V_D, V_B, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_S$
OPVGD(V_{eg}, V_B)	$(V_G, V_D, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S, V_B$
OPVGD(V_{GS}, V_B)	$(V_G, V_D, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_S, V_B$
OPVGD(V_{eg}, V_B, W)	$(V_G, V_D, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{eg}, V_S, V_B$
OPVGD(V_{GS}, V_B, W)	$(V_G, V_D, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_{GS}, V_S, V_B$
OPVGD(V_S, V_B, W)	$(V_G, V_D, V_{th}) \Leftarrow Temp, I_{DS}, W, L, V_S, V_B$

Table 4.5: *Definition of Operators for W Computation.*

Operator	Definition
OPW(V_{eg})	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S$
OPW(V_{GS}, V_G)	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_G$
OPW(V_{GS}, V_S)	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_S$
OPW(V_G, V_S)	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_G, V_D, V_S$
OPW(V_{eg}, V_B)	$(W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S, V_B$
OPW(V_{GS}, V_G, V_B)	$(W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_G, V_B$
OPW(V_{GS}, V_S, V_B)	$(W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_{GS}, V_D, V_S, V_B$
OPW(V_G, V_S, V_B)	$(W, V_{th}) \Leftarrow Temp, I_{DS}, L, V_G, V_D, V_S, V_B$

Table 4.6: *Definition of Operators for I_{DS} Computation.*

Operator	Definition
OPIDS(V_{eg})	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_{eg}, V_D, V_S$
OPIDS(V_{GS}, V_G)	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_{GS}, V_D, V_G$
OPIDS(V_{GS}, V_S)	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_{GS}, V_D, V_S$
OPIDS(V_G, V_S)	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_G, V_D, V_S$
OPIDS(V_{eg}, V_B)	$(I_{DS}, V_{th}) \Leftarrow Temp, W, L, V_{eg}, V_D, V_S, V_B$
OPIDS(V_{GS}, V_G, V_B)	$(I_{DS}, V_{th}) \Leftarrow Temp, W, L, V_{GS}, V_D, V_G, V_B$
OPIDS(V_{GS}, V_S, V_B)	$(I_{DS}, V_{th}) \Leftarrow Temp, W, L, V_{GS}, V_D, V_S, V_B$
OPIDS(V_G, V_S, V_B)	$(I_{DS}, V_{th}) \Leftarrow Temp, W, L, V_G, V_D, V_S, V_B$

4.4 Enhanced MOS Engine

An enhanced architecture for the MOS calculator, initially developed by Laurent de Lamarre in [de Lamarre02], is proposed and is shown in Fig. 4.5. In our proposed architecture, the functions that are kept in the leaf level correspond to the BSIM3V3 model equations. The *standard* sizing and biasing procedures, that existed in the initial implementation [de Lamarre02], were kept in the new architecture. These functions are mainly computing the width in terms of gate-source voltage V_{GS}. Another level was created for the functions, called *elementary API*. These functions compute the width in terms of the overdrive voltage V_{eg}. The uppermost level is dedicated for the sizing and biasing operators that uses all the lower-level procedures and APIs for its implementation. Note that an arrow means that the functions pointed by the arrow head calls the functions at the arrow tail.

The proposed architecture was totally developed in C and integrated into the CAIRO+ framework. It is developed as a standalone library that can be linked to any program in order to compute sizes and biases of a MOS transistor based on the BSIM3V3 transistor model and supporting 0.13μm technology.

4.5 Illustrative example

Let us suppose that the OTA amplifier shown in Fig. 4.6 is to be sized and biased in $0.13\mu m$ technology. The input common mode voltage $V_{IN,CM} = 0.6V$ and the output common mode voltage is $V_{OUT,CM} = 0.6V$. We assume also that the two branches of the amplifier are sized identically, i.e. $W_{M_1} = W_{M_2}$ and $W_{M_3} = W_{M_4}$. We assume also that the number of fingers for all transistors is $M = 1$. Our objective is to implement a sizing procedure for the amplifier using the sizing and biasing operators defined in section 4.3.6. Suppose that the steps to implement the sizing procedure are:

1. We would like to compute the width of the diode-connected transistor M_3 given that:

$$V_{DD} = 1.2V \tag{4.13}$$
$$V_{G,M_3} = V_{D,M_3} = V_{OUT,CM} = 0.6V \tag{4.14}$$
$$V_{S,M_3} = 1.2V \tag{4.15}$$
$$V_{B,M_3} = 1.2V \tag{4.16}$$
$$Temp = 300.15^{\circ}K \tag{4.17}$$
$$L_{M_3} = 2\mu m \tag{4.18}$$
$$I_{DS,M3} = -10\mu A \tag{4.19}$$

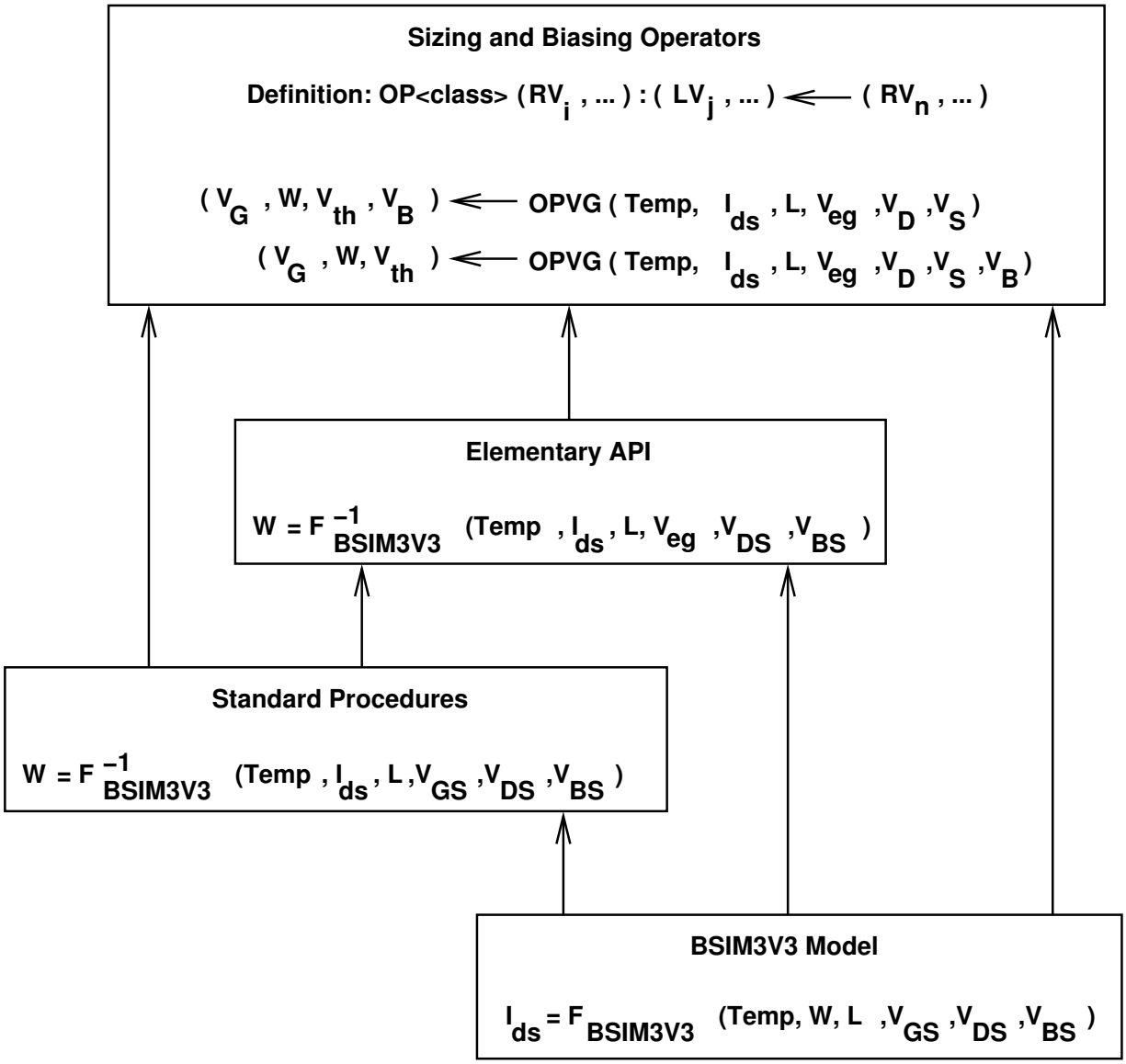

Figure 4.5: *Enhanced Architecture of MOS Engine*.

2. Next, we would like to compute V_{S,M_1} given that:

$$V_{D,M_1} = V_{OUT,CM} = 0.6V \tag{4.20}$$
$$V_{G,M_1} = V_{IN,CM} = 0.6V \tag{4.21}$$
$$Temp = 300.15°K \tag{4.22}$$
$$L_{M_1} = 2\mu m \tag{4.23}$$
$$I_{DS,M1} = 10\mu A \tag{4.24}$$
$$V_{eg,M1} = 0.12V \tag{4.25}$$
$$V_{B,M_1} = 0V \tag{4.26}$$

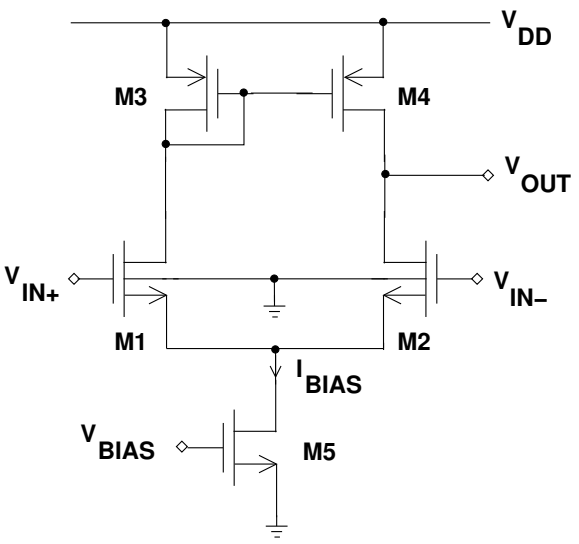

Figure 4.6: *Single stage OTA amplifier*.

3. Finally, we would like to compute the bias V_{G,M_5} given that:

$$\begin{aligned} V_{D,M_5} &= V_{S,M_1} && (4.27)\\ Temp &= 300.15^{\circ}K && (4.28)\\ L_{M_5} &= 2\mu m && (4.29)\\ I_{DS,M5} &= 20\mu A && (4.30)\\ V_{eg,M5} &= 0.12V && (4.31)\\ V_{S,M_5} &= 0V && (4.32)\\ V_{B,M_5} &= 0V && (4.33) \end{aligned}$$

To implement the above sizing procedure, we perform the following calls to the sizing and biasing operators:

- To compute W_{M_3} based on the known parameters in step 1, we call:

$$OPW(V_G, V_S) : (W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_G, V_D, V_S \quad (4.34)$$

We get

$$\begin{aligned} OPW : (W_{M_3}, V_{B,M_3}, V_{th,M_3}) &\Leftarrow Temp, I_{DS,M_3}, L_{M_3}, V_{G,M_3}, V_{D,M_3}, V_{S,M_3} && (4.35)\\ &\Rightarrow (7.75976\mu m, 1.2V, -0.335232V) && (4.36) \end{aligned}$$

- To compute V_{S,M_1} based on the known parameters in step 2, we call:

$$OPVS(V_{eg}, V_B) : (V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G, V_B \quad (4.37)$$

We get

$$OPVS : (V_{S,M_1}, V_{th,M_1}, W_{M_1}) \Leftarrow Temp, I_{DS,M_1}, L_{M_1}, V_{eg,M_1}, V_{D,M_1}, V_{G,M_1}, V_{B,M_1} \quad (4.38)$$

$$\Rightarrow (0.155313V, 0.324687V, 8.17438\mu m) \quad (4.39)$$

- To compute V_{G,M_5} based on the known parameters in step 3, we call:

$$OPVG(V_{eg}) : (V_G, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S \quad (4.40)$$

We get

$$OPVG : (V_{G,M_5}, V_{B,M_5}, V_{th,M_5}, W_{M_5}) \Leftarrow Temp, I_{DS,M_5}, L_{M_5}, V_{eg,M_5}, V_{D,M_5}, V_{S,M_5} \quad (4.41)$$

$$\Rightarrow (0.412767V, 0V, 0.292767V, 17.1798\mu m) \quad (4.42)$$

In Table 4.7, the sizing and biasing results agree with simulation. The amplifier was sized, biased and simulated in $0.13\mu m$ CMOS technology. We conclude that sizing procedures can be written as a sequence of sizing and biasing operators. One of our main objectives is to automatically generating this sequence based on the circuit topology and designer's hypotheses. This will be presented in the rest of the chapters along with many other objectives.

Table 4.7: *Synthesis vs Simulation Results..*

Parameter	Synthesis			Simulation		
	M_1	M_3	M_5	M_1	M_3	M_5
$I_{DS}(\mu A)$	10.0	-10.0	20.0	10.0	-10.0	20.0
$V_{GS}(V)$	0.444687	-0.6	0.412767	0.44469	-0.6	0.41277
$V_{DS}(V)$	0.444687	-0.6	0.155313	0.44469	-0.6	0.15531
$V_{BS}(V)$	-0.155313	0.0	0.0	-0.15531	0.0	0.0
$V_{th}(V)$	0.324687	-0.335232	0.292767	0.32469	-0.33523	0.29277
$V_{eg}(V)$	0.12	-0.264768	0.12	0.12	-0.26477	0.12
$g_m(mA/V)$	0.136233	0.0701239	0.2641	0.13623	0.070124	0.26410
$g_{ds}(\mu A/V)$	0.714603	0.255633	18.7045	0.71460	0.25563	18.705
$g_{mb}(mA/V)$	0.0272121	0.0146514	0.0566178	0.027212	0.014651	0.056618
$C_{gd}(fF)$	4.17931	3.29543	13.4681	4.1793	3.2954	13.468
$C_{gs}(pF)$	0.124526	0.126692	0.262478	0.12452	0.12669	0.26247
$C_{sd}(fF)$	0.0740517	0.101938	4.26634	0.074052	0.10194	4.2664
$C_{bd}(fF)$	0.0619098	0.0779054	3.68848	0.061910	0.077905	3.6885

4.6 Conclusions

In this chapter, a method for accurately sizing and biasing a MOS transistor is proposed. The method takes into account the second-order effects that may introduce some inaccuracy during hand calculations. The method relies on the definition of a set of sizing and biasing operators that interfaces to an accurate standard BSIM3V3 compact model. The operators represent the interface of our reuse strategy to the target technology. The proposed set of operators provides a unified formulation that accounts for both operating-point-driven formulation and standard simulation approach. A new architecture for the MOS calculator is proposed to smoothly integrate the operators into the CAIRO+ framework in the form of a standalone library. This library can be linked to any synthesis system to accurately size and bias MOS transistors. The operators can be easily extended to account for other model levels (such as BSIM4, PSP, ...) or uncommon interconnections (such as bulk input connections in [Chatterjee05]). Currently the operators support the BSIM3V3 compact model in the 0.13μ technology.

Chapter 5

Device Sizing and Biasing Methodology

5.1 Introduction

In the previous chapter, the transistor sizing and biasing methodology was presented to size and bias a single transistor. We extend the ideas to develop more complex analog basic building blocks called *devices*. We show also how design plans for devices are represented using *device dependency graphs*. This is considered a necessary step to deal with hierarchy in analog design.

In section 5.2, we discuss how hierarchy is represented in CAIRO+.

In section 5.3, we present definitions for our basic building block called *devices*.

In section 5.4, we show how design plans for devices are represented using *device dependency graphs*.

In section 5.5, we illustrate an example for device sizing and biasing.

Finally, the chapter is concluded in section 5.6.

5.2 Hierarchy in Analog Design

The hierarchy in analog design has been always a difficult problem that is still unsolved. Generally, hierarchy is needed to decompose a problem into more tractable sub-problems. Today, analog designers rely on analyzing flattened analog circuits. This is because traditional analysis techniques such as DC and AC analysis assume that the netlist is flattened. Consequently, information about hierarchy is not preserved during traditional circuit analysis.

In the past, some attempts have been made to define hierarchy. For example, OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a] represents a substantial departure from the optimization style of analog synthesis. OASYS seeks to formalize the mechanisms used by human designers and capture this expertize in repeatable form. To accomplish this, OASYS relied on three key organizational principles:

1. **Hierarchy:** The decomposition of a large problem into a small number of simpler problems. In OASYS, circuits are decomposed into smaller sub-circuits, and these subcircuits are them-

selves successively decomposed. Eventually, transistors and other fundamental components are designed explicitly and the completed circuit re-assembled.

2. **Style Selection:** The selection of an interconnection of subcircuits (a topology where each component may itself be decomposed hierarchically) to best satisfy a particular given set of performance specifications.

3. **Translation:** The mathematical mapping of performance specifications for a particular circuit in a design hierarchy into sets of specifications for each of its component subcircuits.

In OASYS, each topology has one design plan associated with it, and this plan is a linear sequence of executable steps that are created by experienced designers, called plan steps. Plan steps may perform *design heuristics*, *computations* of currents and voltages needed to proceed, or by successively *refining* the design through invocation of the selection and translation mechanisms for a lower level sub block, once sufficient information has been deduced by the current plan.

In CAIRO+ framework [Dessouky01, Tuong06], an analog IP [Lopez05] is considered as an abstract hierarchy of subcircuits. A subcircuit is an unsized schematic view. Leaf subcircuits are called *devices* and higher-level subcircuits are called *modules*. For each subcircuit, the designer creates a parameterized generator in CAIRO+ language. A parameterized generator is an instantiable object that receives design and technology parameters and provides behavioral, structural and physical views.

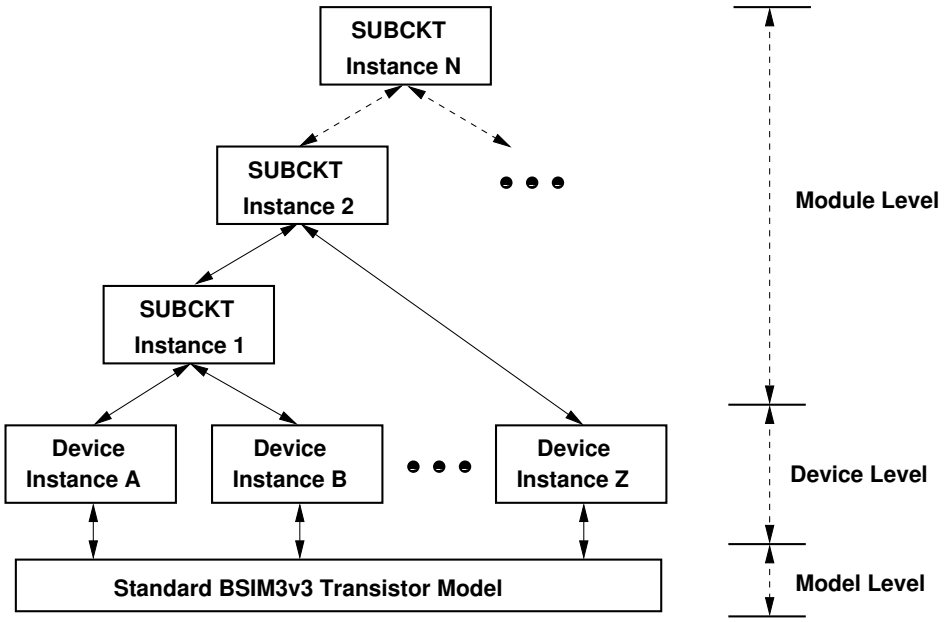

Figure 5.1: *Hierarchical instantiation tree and parameter exchange.*

Following these definitions, an instance of the analog IP is viewed as a hierarchical instantiation tree of device and module generators. Higher-level module generators instantiate lower-level

module and device generators. A device-level MOS generator interfaces directly to the lowest hierarchical level which is a standard BSIM3v3 transistor model. The generator in one level in the hierarchy exchanges electrical parameters with its direct ascendants and descendants only. Fig. 5.1 illustrates an example of a hierarchical instantiation tree and parameter exchange in CAIRO+. In this tree, device instances *A*, *B*, ... , and *Z* call the standard BSIM3v3 transistor model. The higher-level module instance *SUBCKT 2* instantiates both the lower-level module instance *SUBCKT 1* and the device instance *Z*. *SUBCKT 2* instance exchanges parameters with *SUBCKT 1*, device instance *Z* and *SUBCKT N*.

A device represents the building block for the design construction in CAIRO+. Fig. 5.2 show many possible device examples. A device may represent simple passive components like a resistor (A) or a capacitor (B). It may represent matched component pairs such as matched resistors (C and E) and matched capacitors (D and F). It may represent also active components like a MOS transistor. Or, it represents a complex device comprising of more than one transistor like a differential pair (G), a current mirror (H), a group of transistors sharing the same gate (I), etc.

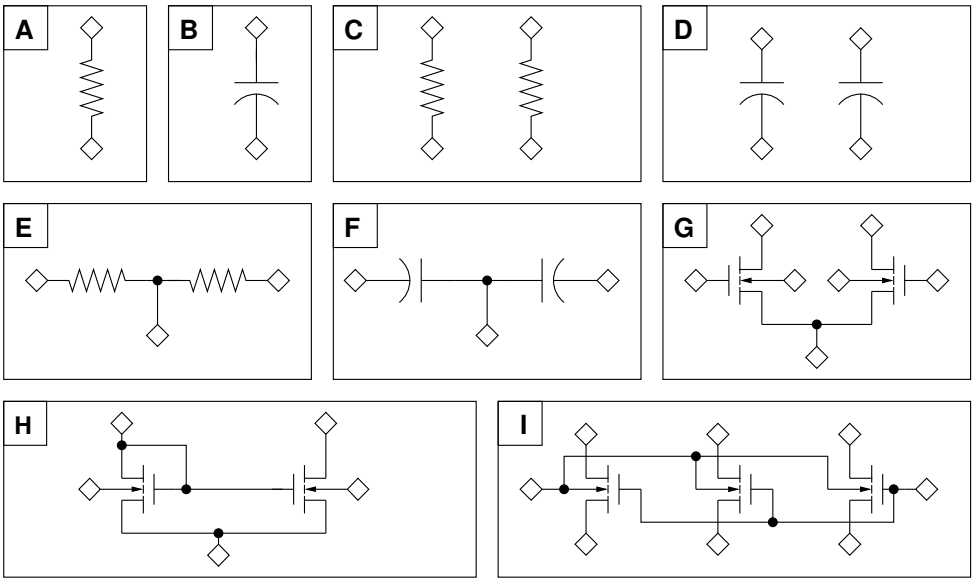

Figure 5.2: *Low-level devices*.

Our main aim is to define a suitable hierarchical representation for analog circuits. This is accomplished by defining and elaborating the device in order to serve as a constitutive and reusable building block. The work done by Mohamed Dessouky [Dessouky01] and Vincent Bourguet [Bourguet04] elaborated the physical representation of a device. In this chapter, the electrical representation of the device that describes its electrical behavior is elaborated. New concepts are defined for devices so these can be synthesized by the CAIRO+ framework. Synthesizing a device means generating a suitable *design plan*, i.e. sizing procedure, for that device. The sizing procedure represents a reusable knowledge about the device that can be inherited by higher level modules in order to size and bias that device. To represent the design plan for a device, an intermediate

representation called *device dependency graphs* is proposed.

5.3 Device Definition

5.3.1 The Transistor Packing

A *device* is defined as one or more transistors packed together as one atomic building block. The following conditions should be respected during transistor packing:

1. Any set of transistors that form one distinct electrical function should be packed together.

2. Any set of transistors that either share a subset of electrical parameters such as W, L, V_{GS}, or are matched in the physical level, should be packed together.

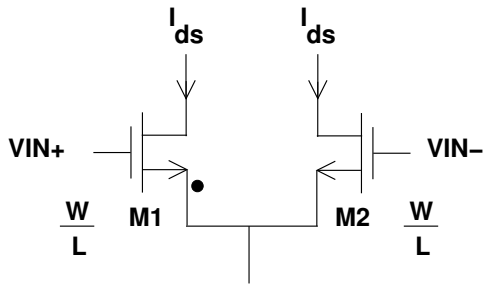

Figure 5.3: *Transistor packing for a differential pair*.

As shown in Fig. 5.3, transistors M_1 and M_2 should be packed together as:

1. They form one atomic and complete function of a differential pair.

2. They share the same W and L, and should be physically matched to reduce current mismatch. Consequently, both M_1 and M_2 are inter-digitized in Fig. 5.4.

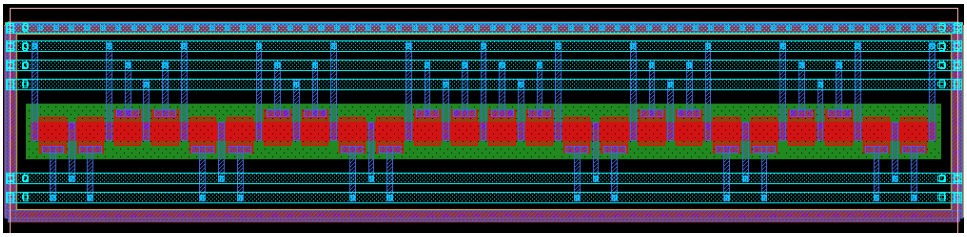

Figure 5.4: *Inter-digitization of M_1 and M_2.*

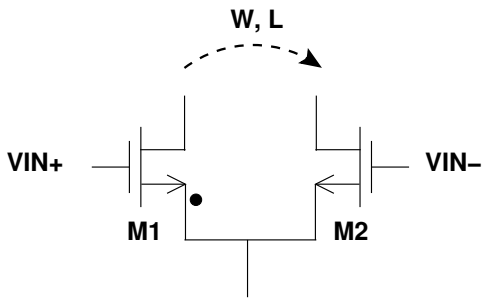

Figure 5.5: *Parameter propagation in a differential pair*.

5.3.2 The Reference Transistor

In the traditional method for analog design, the designer sizes at first a minimum set of primary transistors. Their sizes are then copied into some other secondary transistors in the circuit. These design steps are followed, for example, in the half-circuit analysis where one half of the circuit is sized the other inherits the same sizes of the first half. To mimic these design principles in an automated flow, the concept of *the reference transistor* is proposed. Each device contains only one primary transistor, namely the reference transistor. The reference transistor is first sized and biased. Then, its electrical parameters are *propagated* to the other transistors in the device. Sizing and biasing a device means simply sizing and biasing the reference transistor and propagating electrical parameters to the other transistors, if any. Reference transistors are marked by a dot as shown in Fig. 5.3.

For example, in the differential pair shown in Fig. 5.5, one might possibly select M_1 to be the reference transistor, and M_2 to be the secondary one. It is said that M_1 propagates the width and length to M_2.

5.3.3 Sizing and Biasing Operators Declaration

In order to size and bias a reference transistor in a device, the device links with the sizing and biasing operator classes it needs. The choice of operators depends mainly on the interconnection of the reference transistor. For example, a diode-connected reference transistor is sized and biased using the operator class *OPVGD*. Therefore, a device like the current mirror having its reference transistor diode-connected should declare *OPVGD* as one of its linked operators.

5.3.4 Device Constraints

It is essential for a device to declare necessary and sufficient constraints that ensure its proper sizing and operation. Different types of constraints can be defined into a device e.g. *functional* and *robustness* constraints as in *the sizing rules method*[Graeb01]. Constraints are viewed as a way to propagate parameters along the hierarchy. Here, two types of constraints are proposed:

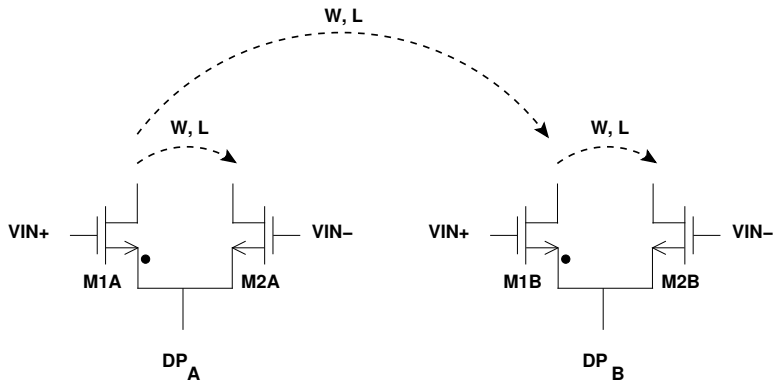

Figure 5.6: *Parameter propagation using constraints*.

1. **Intrinsic Constraints:** These constraints are introduced inside a device. Their main purpose is to propagate parameters from the reference transistor to secondary ones. For example, in Fig. 5.6, W and L are propagated inside DP_A and DP_B, from M_{1A} to M_{2A} and from M_{1B} to M_{2B} respectively. Mathematically, intrinsic constraints are expressed as *linear equality constraints*,
$$\left[P^{tr}_{elec}\right]_{N\times 1} = \left[K\right]_{N\times M} \cdot \left[P^{tr}_{elec,ref}\right]_{M\times 1} \tag{5.1}$$
where P^{tr}_{elec} is a matrix of the electrical parameters of all secondary transistors, K is a matrix of constants and $P^{tr}_{elec,ref}$ is a matrix of the electrical parameters of the reference transistor. Applying this to the differential pair DP_A, one gets
$$\begin{bmatrix} \mathrm{W_{2A}} \\ \mathrm{L_{2A}} \end{bmatrix}_{2\times 1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2\times 2} \cdot \begin{bmatrix} \mathrm{W_{1A}} \\ \mathrm{L_{1A}} \end{bmatrix}_{2\times 1} \tag{5.2}$$
2. **Extrinsic Constraints:** These constraints are propagated from a reference device to other secondary devices. For instance, in fully-differential circuits, half of the circuit is sized and the other half is made identical of the sized half. To understand this, assume that in Fig. 5.6, the differential pair DP_A is to be first sized. Then its sizes are to be copied to another identical differential pair DP_B. In this case, the width and length are propagated from the reference device DP_A to the secondary device DP_B as shown in the figure. One can also imagine that this propagation is done between reference transistors corresponding to different devices. Mathematically, extrinsic constraints are expressed as *linear equality constraints*,
$$\left[P^{subckt}_{elec}\right]_{N\times 1} = \left[K\right]_{N\times M} \cdot \left[P^{subckt}_{elec,ref}\right]_{M\times 1} \tag{5.3}$$
where P^{subckt}_{elec} is a matrix of the electrical parameters of similar devices that need to be sized and biased, K is a matrix of constants and $P^{subckt}_{elec,ref}$ is a matrix of the electrical parameters of the sized and biased reference device. Applying this to the differential pair DP_B, one gets
$$\begin{bmatrix} \mathrm{W_{1B}} \\ \mathrm{L_{1B}} \end{bmatrix}_{2\times 1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2\times 2} \cdot \begin{bmatrix} \mathrm{W_{1A}} \\ \mathrm{L_{1A}} \end{bmatrix}_{2\times 1} \tag{5.4}$$

5.3.5 External Device Connectors

Each device has a set of external connectors, connected to its external inputs and outputs. For example, the differential pair shown in Fig 5.7 has connector g_1 as its positive input terminal, g_2 as its negative input terminal, b as the common bulk, s as the common source, d_1 and d_2 as the drain terminals. In general, the external connectors are created at device instantiation and cannot be altered afterwards. As an exception, some parameters may control the creation of a common bulk terminal as in Fig. 5.7 or consider the bulk is connected to the source for each separate transistor .

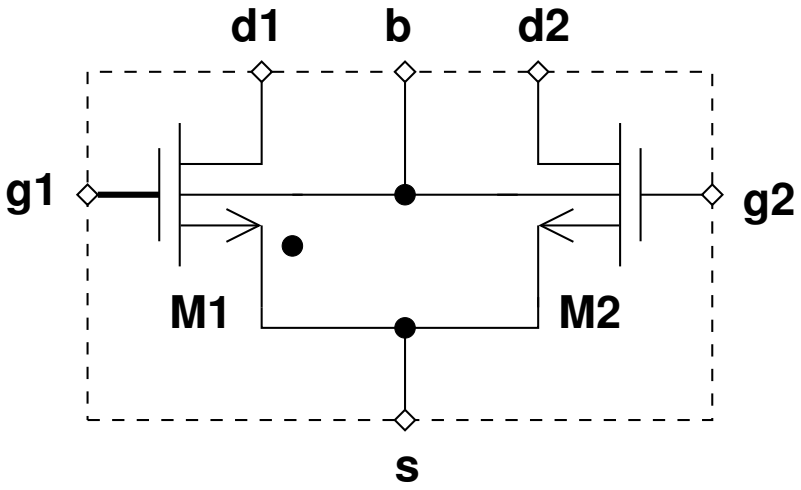

Figure 5.7: *External connectors for a differential pair*.

5.4 Device Dependency Graphs

5.4.1 Device Parameters Revisited

Recall from section 4.3, the quadratic model of the NMOS transistor at ambient temperature $Temp$ in strong inversion is:

$$I_{DS,n} = \frac{\mu_n}{2} C_{ox} \frac{W}{L} (V_{GS,n} - V_{th,n})^2 (1 + \lambda_n V_{DS,n}) \tag{5.5}$$

$$= \frac{\mu_n}{2} C_{ox} \frac{W}{L} (V_{eg,n})^2 (1 + \lambda_n V_{DS,n}) \tag{5.6}$$

$$V_{th,n} = V_{th0,n} + \gamma_n (\sqrt{2\phi_F - V_{BS,n}} - \sqrt{2\phi_F}) \tag{5.7}$$

$$\phi_F = \frac{k \cdot Temp}{q} \ln \frac{N_A}{n_i} \tag{5.8}$$

From our previous discussion in section 4.3, the design parameters $Temp$, I_{DS}, L, W, V_{eg} or V_{GS}, V_{BS} and V_{DS} have been chosen. In our proposed method, we are interested in potentials rather than potential differences. This transforms our design parameters to $Temp$, I_{DS}, L, W, V_{eg} or V_{GS}, V_G, V_S, V_D and V_B. Let us examine how each of these parameters is set:

1. The temperature $Temp$ is set by the designer to set the operating temperature.
2. Transistor length L is set by the designer in both standard simulation approach and operating point-driven formulation.
3. Transistor width W is set by the designer in standard simulation approach.
4. The drain-source current I_{DS} is set by the designer in the operating-point-driven formulation. In the standard simulation approach, I_{DS} is computed from BSIM3V3 model equations.
5. The overdrive voltage V_{eg} and the gate-source voltage V_{GS} are set by the designer in the operating-point-driven formulation.
6. The bulk potential V_B is set by the designer in the operating-point-driven formulation if the transistor is not bulk-source connected. If the transistor is bulk-source connected, the bulk potential V_B is made equal to the source potential V_S.

For the terminal potentials V_G, V_S and V_D, the BSIM3V3 model equations will be inverted using the principles presented in subsection 4.3.1. This occurs in both the standard simulation approach and the operating-point-driven formulation.

5.4.2 Dependency Graph Definition

In order to be able to automatically generate suitable sizing procedures for devices, an intermediate representation has been identified and is called *device dependency graphs*. We start by defining nodes and arcs of the graph and how the nodes are related to each other in the graph.

5.4.2.1 Node Definition

A *node* represents an electrical parameter, namely $Temp$, I_{DS}, L, W, V_{eg}, V_{GS}, V_G, V_S, V_D or V_B. Each graph node possesses some properties:

1. It has an electrical type: *drain-source current*, *length*, *width*, *parameter*, *connector*, *temperature*, *gate voltage*, ... , etc.
2. It has a list of multiples names , called *aliases*. Aliases are used to designate *equivalent* electrical parameters. For example, if $W_1 = W_2$ then a node of type *width* will be created which will have the aliases W_1 and W_2. This simply means that W_1 is equivalent to W_2.

5.4.2.2 Arc Definition

An *arc* represents a weighted dependence of one node v on another node u. Therefore, an arc is *directed* since it has a propagation direction from node u to node v. It is said that v depends on u. This is formally written as $v \xleftarrow{w_i} u$. It means also that $v = w_i \times u$.

5.4.2.3 Dependency Rule Definition

A *dependency rule* expresses electrical dependencies of a node v on other nodes $u_1, u_2, \cdots, u_n$. This is formally written as $v \overset{w_i}{\longleftarrow} u_1, u_2, \cdots, u_n$. Multiple nodes may be computed from the rule simultaneously. This is expressed as $v_1, v_2, \cdots, v_m \overset{w_i}{\longleftarrow} u_1, u_2, \cdots, u_n$. A dependency rule possesses some properties:

1. It has a rule type: *constraint* or *operator*.
2. It has a constant weight. If it is omitted, it has a default value of 1.0.
3. It has a list of affected nodes. These are the nodes computed by applying the rule.
4. It has a list of affecting nodes. These are the nodes that are used to compute an affected node.
5. It has a name of an operator to execute in order to compute an affected node.

To illustrate an example of a dependency rule of type *constraint*, consider that the widths W_2 and W_3 are imposed using the constraint $W_2 = W_3 = 5 \times W_1$. This is formally written as $W_2, W_3 \overset{5}{\leftarrow} W_1$. This can be represented by the dependency graph shown in Fig. 5.8. The constraint has a

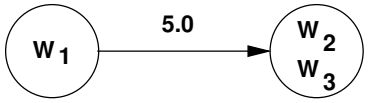

Figure 5.8: *Dependency graph for* $W_2, W_3 \overset{5}{\leftarrow} W_1$.

constant weight of 5.0. W_1 is the affecting node. The affected node has the aliases (W_2, W_3). The constraint has no operator to execute.

To illustrate another example of a dependency rule of type *operator*, consider the operator *OPVGD* generated for a diode-connected and bulk-source connected reference transistor of a current mirror

$$OPVGD(V_{eg}) : (V_G, V_D, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S \tag{5.9}$$

The operator has a default weight of 1.0. The list of affected nodes is (V_G,V_D,V_B,V_{th},W). The list of affecting nodes is ($Temp$,I_{DS},L,V_{eg},V_S). The operator name to execute is $OPVGD(V_{eg})$. The graph representation of the operator *OPVGD* is shown in Fig. 5.9. The nodes on the left hand-side are the affecting nodes. The nodes on the right hand-side are the affected or computed nodes. Each effected node is labelled with the name of the operator used to compute it. In our case, it is *OPVGD* for both affected nodes. Each arrow points to the direction of parameter dependency (or propagation).

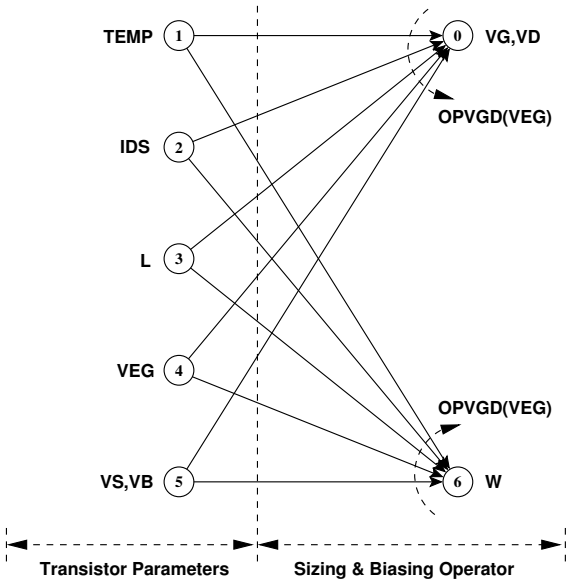

Figure 5.9: *Graph representation for the operator* $OPVGD(V_{eg})$.

5.4.3 Constructing Complex Dependency Graphs

Suppose that the current mirror, shown in Fig.5.10, is to be sized while respecting a mirror ratio of 1:5. The mirror ratio is ensured by imposing the constraint $W_2 = W_3 = 5 \times W_1$. The resulting

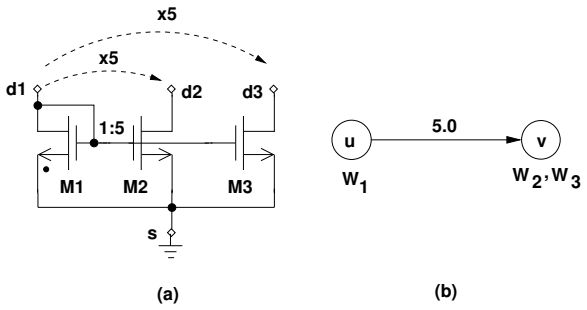

Figure 5.10: *Current mirror: (A) Device constraints on widths, (b) Parameter propagation*.

dependency graph of the current mirror is shown in Fig. 5.11. First, a new hierarchical level has been created for the current mirror as a standalone device. This is shown as the first column in the graph, labelled *device parameters*. The role of this level is to receive the parameters of the current mirror and propagate them to its internal transistors in the *transistor level* shown in the second column. In the second column, mainly the reference transistor M_1 is biased. Hence, it appears on all nodes. Since constraints have been imposed to propagate V_{eg}, $Temp$ and L from M_1 to M_2 and M_3, all share nodes 1, 2 and 4. In node 5, V_S and V_B share the same node since M_1 is assumed bulk-source connected. The parameters V_{G,M_1}, V_{D,M_1} and W_{M_1} are computed in the third column. Recall the constant current ratio of 1:5 that is imposed on the current mirror. This

constraint appears in the last column, where W_{M_1} in node 6 is weighted by 5.0 and propagated to W_{M_2} and W_{M_3} in node 7. Note also, how connectors have been added to appropriate nodes in the device dependency graph.

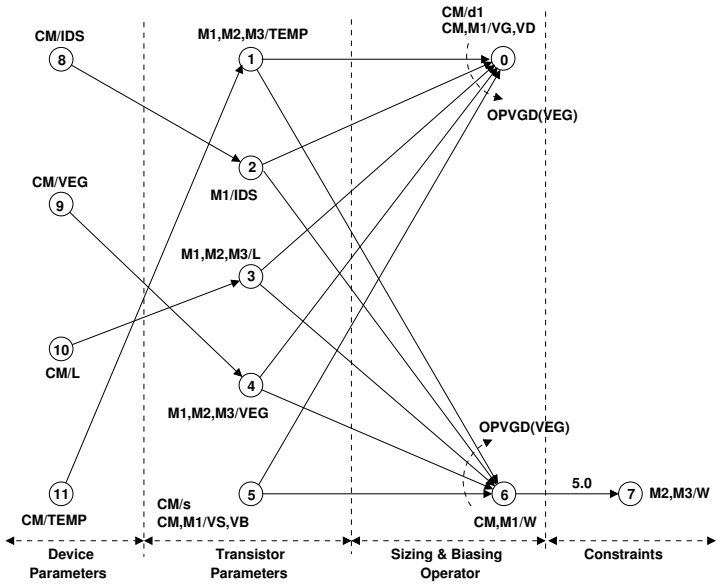

Figure 5.11: *Dependency Graph of the current mirror with width constraint. Assuming ideal current mirror.*

The most important to consider about constraints is that they are guaranteed to be satisfied by construction since they make part of the device dependency graph.

5.5 Illustrative example

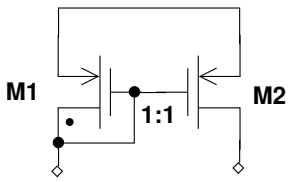

Figure 5.12: *A simple current mirror.*

Let us assume that the simple current mirror of Fig. 5.12 is to be sized and biased in $0.13\mu m$ technology. We choose to set the device parameters $Temp$, I_{DS}, V_{eg}, L and V_S (= V_B). The resulting dependency graph of the simple current mirror is shown in Fig. 5.13.

Since the reference transistor M_1 is diode-connected, the operator

$$OPVGD(V_{eg}) : (V_G, V_D, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S \tag{5.10}$$

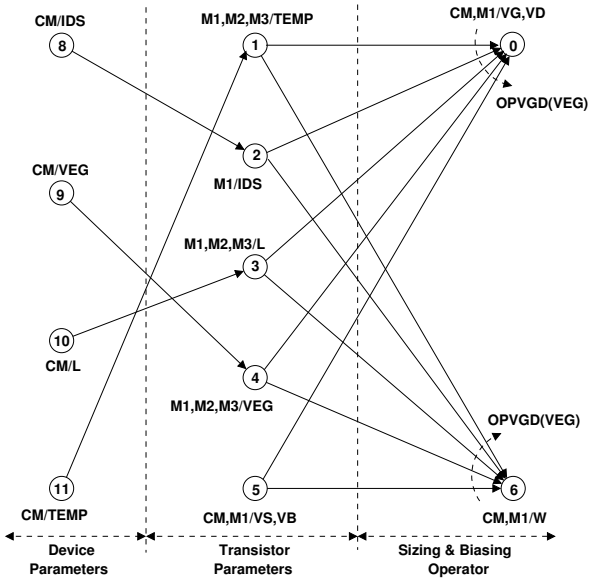

Figure 5.13: *Dependency graph of the simple current mirror.*

is called to compute the parameters V_{G,M_1}, V_{D,M_1}, V_{B,M_1}, V_{th,M_1} and W_{M_1}. Note that $W_{M_2} = W_{M_1}$. Assuming that the number of fingers $M = 1$ and that:

$$V_{DD} = 1.2V \tag{5.11}$$
$$V_{eg,M_1} = -0.12V \tag{5.12}$$
$$V_{S,M_1} = 1.2V \tag{5.13}$$
$$V_{B,M_1} = V_{S,M_1} \tag{5.14}$$
$$Temp = 300.15°K \tag{5.15}$$
$$L_{M_1} = L_{M_2} = 2\mu m \tag{5.16}$$
$$I_{DS,M1} = -10\mu A \tag{5.17}$$
$$W_{M_2} = W_{M_1} \tag{5.18}$$

We get

$$OPVGD : (V_{G,M_1}, V_{D,M_1}, V_{B,M_1}, V_{th,M_1}, W_{M_1}) \Leftarrow Temp, I_{DS,M_1}, L_{M_1}, V_{eg,M_1}, V_{S,M_1} \tag{5.19}$$
$$\Rightarrow (0.740562, 0.740562, 1.2, -0.339438, 30.8054\mu m) \tag{5.20}$$

In Table 5.1, the sizing and biasing results of the simple current mirror agree with simulation.

5.6 Conclusion

In this chapter, a methodology is proposed to introduce and refine the hierarchy for analog design. The concepts developed in this chapter have been used to automatically generate sizing

Table 5.1: *Synthesis vs Simulation Results..*

Parameter	Synthesis		Simulation	
	M_1	M_2	M_1	M_2
$I_{DS}(\mu A)$	-10.0	-10.0	-10.0	-10.0
$V_{GS}(V)$	-0.459438	-0.459438	-0.45944	-0.45944
$V_{DS}(V)$	-0.459438	-0.459438	-0.45944	-0.45944
$V_{BS}(V)$	0.0	0.0	0.0	0.0
$V_{th}(V)$	-0.339438	-0.339438	-0.33944	-0.33944
$V_{eg}(V)$	-0.12	-0.12	-0.12	-0.12
$g_m(mA/V)$	0.131655	0.131655	0.13165	0.13165
$g_{ds}(\mu A/V)$	0.342102	0.342102	0.34210	0.34210
$g_{mb}(mA/V)$	0.0295237	0.0295237	0.029524	0.029524
$C_{gd}(fF)$	12.9712	12.9712	12.971	12.971
$C_{gs}(pF)$	0.434976	0.434976	0.43493	0.43493
$C_{sd}(fF)$	0.315307	0.315307	0.31531	0.31531
$C_{bd}(fF)$	0.23738	0.23738	0.23738	0.23738

and biasing procedures for our elementary reusable building blocks, called *devices*. An *Application Program Interface* (API) has been identified as shown in appendix B and integrated as part of the language CAIRO+. The API is used to declare information about the structure and functionality of the device. The method has been successfully used to implement devices such as a transistor, a differential pair, a current mirror and an array of transistors.

Chapter 6

Circuit Sizing and Biasing Methodology

6.1 Introduction

Nowadays, researchers focus on developing methods and tools to manage and deploy analog intellectual property cores (IP). In current approaches[Porte08, Stefanovic03, Stefanovic07, Stefanovic05], knowledge is fully determined by the designer's expertise and coded using high-level languages such as C, MATLAB [MathWorks], Verilog-AMS [Verilog-AMS], VHDL-AMS [VHDL-AMS] or SystemC-AMS [SystemC-AMS]. Contemporary EDA tools hardly ensure knowledge consistency. The fundamental reason behind this is the lack of intermediate representations that can represent, judge and correct analog design knowledge. Capitalizing knowledge will be the main driving force in the next coming years. Therefore, it is important to develop tools that ensures knowledge consistency. In this context, we propose a methodology for automatic generation of suitable design plans for circuit sizing and biasing while respecting designer hypotheses and ensuring consistency of knowledge stored in design plans.

In the previous chapter, we focused on defining basic building blocks called *devices*. In this chapter, analog circuits are constructed by instantiating devices and interconnecting them. Then, a suitable design plan is automatically extracted for the circuit topology based on designer hypotheses and performance specifications. Since a design plan represents a consistent knowledge about the circuit, it should not contain any inconsistency. Inconsistency appears as *redundant*, *cyclic* or *conflicting* hypotheses [Wu94]. Our proposed methodology mainly focuses on automatically identifying and resolving these forms of inconsistency. We show that redundant hypotheses depends on how knowledge is expressed. Moreover, an under-specified design results from cyclic hypotheses. In addition, an over-specified design results from conflicting hypotheses. We show that many aspects in analog design results into such inconsistencies. Identifying sources of inconsistency is essential in representing knowledge, judging its effectiveness, and developing resolution techniques ensuring its consistency. This acquires EDA tools lots of insight to manipulate analog design knowledge knowledge efficiently.

In our proposed methodology, the analog circuit is defined as a hierarchy of subcircuits. Leaf

subcircuits are called *devices* and higher-level subcircuits are called *modules*. Each subcircuit is represented by dependency graph. A dependency graph expresses electrical dependencies of subcircuit DC parameters on the designer's selected parameters. The dependency graph of the analog circuit is constructed in a hierarchical bottom-up approach. This is performed by merging dependency graphs for children devices and lower-level modules. Generally, the resulting graph contains directed cycles. To represent a design plan, the graph should be a directed acyclic graph (DAG). Our method detects and removes existing directed cycles from the dependency graph, thus obtaining the DAG which defines the design plan. The design plan is executed in a top-down fashion in order to compute the DC operating point and the widths of all transistors. In this chapter, we present the algorithms developed to implement our proposed methodology.

In section 6.2, we give a brief introduction about hierarchy in CAIRO+.

In section 6.3, we describe how to represent design plans for circuits using *module dependency graphs*.

In section 6.4, we present a bottom-up construction methodology to construct module dependency graphs.

In section 6.5, we show how our proposed methodology deals with different aspects in analog design.

In section 6.6, we discuss how to evaluate module dependency graphs in a top-down fashion.

In section 6.7, we present the general synthesis routine based on the principles presented in previous sections.

In section 6.8, we present a detailed example demonstrating the ideas and principles of this chapter.

Finally, we conclude the chapter in section 6.9.

6.2 Hierarchy in CAIRO+

A discussed in section 5.2, an analog circuit is constructed as a hierarchy of interconnected subcircuits. *Leaf* subcircuits are called *devices* and higher-level subcircuits are called *modules*. Higher-level modules instantiate devices and lower-level modules. In general, a module represents a level of design abstraction in the hierarchy. The higher the module is, the higher the level of design abstraction. The design abstraction is determined by the nature of knowledge and the type of input/output parameters.

Since each module has its input/output parameters, modules can communicate with each other. The communication is performed using a predefined mechanism between successive levels. This is illustrated in Fig. 6.1. In the figure, the current module level *N*, reads an input parameter from the higher module level *N+1* using a *GET_VALUE* and sets back the output parameter to the higher module level using a *SET_VALUE*. Similarly, the current module level *N* sets an input parameter in the lower module level *N-1* using a *SET_PARAM* and reads an output parameter

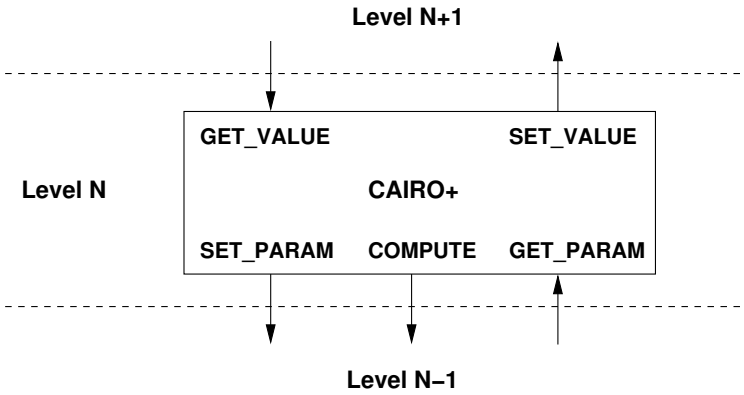

Figure 6.1: *Communication mechanism between successive hierarchical levels*.

of the lower module level *N-1* using a *GET_PARAM*. The current module level *N* can execute a procedure residing in the lower module level *N-1* using a *COMPUTE*.

6.3 Module Dependency Graphs Definition

A design plan represents consistent knowledge about an analog circuit. Since we aim at automatically generating suitable design plans for analog IPs, we use dependency graphs to represent analog design knowledge. In this section, we define *dependency graphs* for modules. In subsequent sections, we show how to automatically generate and evaluate *module dependency graphs*.

6.3.1 Module Parameter Revisited

A module level represent a distinct level of abstraction. It may represent system, functional, block or circuit levels [Doboli03, Jancke06, Martens08]. Therefore, a module level defines its own set of input and output parameters. The type of parameter depends on its role in its level of design abstraction. For instance, a biasing current in the circuit level can be determined from a specification on the unity-gain frequency in the system level, as depicted in Fig. 2.8. Therefore, the unity-gain frequency is considered as an input parameter to the circuit level. While the biasing current is considered as an output parameter of the circuit level. It is computed in the circuit level and returned back to the system level.

6.3.2 Dependency Graph Definition

6.3.2.1 Node Definition

A *node* can be one of the following types:

1. An electrical parameter, namely $Temp$, I_{DS}, L, W, V_{eg}, V_{GS}, V_G, V_S, V_D or V_B.

2. An input parameter to the module. The input parameter nodes are the root nodes that do not have incident arcs. For example, in Fig. 6.2, $AMPLIFIER/V_{eg,CM}$ is an input parameter for the amplifier module.

3. A propagation parameter, called *link parameter*, used to propagate electrical information from one node to another. For example, the node veg in Fig. 6.2 propagates the module input parameter $AMPLIFIER/V_{eg,CM}$ to the current mirror.

4. An input parameter to a device. As described in section 5.4.3, these parameters belong to a device and propagate parameter values received by the device to its internal transistors. For example, in Fig. 6.2 $CURRENT\_MIRROR/V_{eg}$ is an input parameter for the current mirror.

5. An input parameter of a transistor belonging to a device. For example, M_1/V_{eg} is an input parameter of the reference transistor M_1 of the current mirror.

6. An output parameter from a designer-defined procedure. The designer-defined procedure (DDP) describes the dependency of one output parameter on its input parameters. DDPs will be described in section 6.3.2.3.

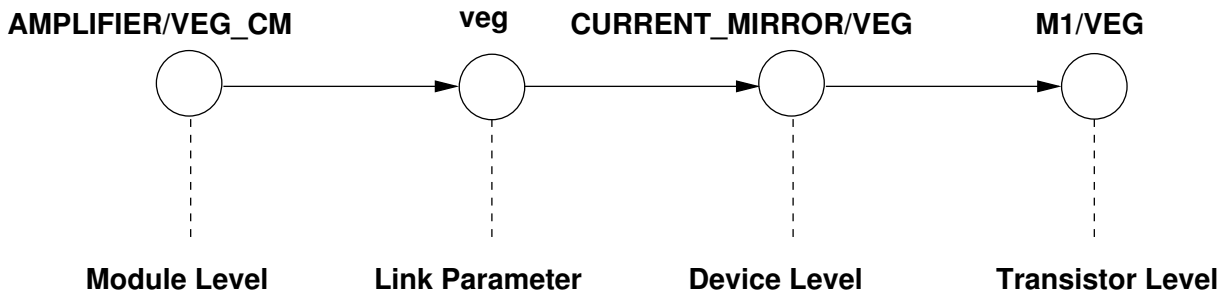

Figure 6.2: *Different node types*.

To summarize Fig. 6.2: *VEG_CM* is an input parameter of module *AMPLIFIER*. It is propagated to the link parameter *veg*. The link parameter *veg* propagates its value to the parameter *VEG* of the device *CM*. Finally, the device *CM* copies the parameter value into the reference transistor M_1.

Formally, each node has some properties:

1. It has an electrical type: *drain-source current*, *length*, *width*, *parameter*, *connector*, *temperature*, *gate voltage*, *link*, ... etc. The link parameter type does not exist for devices (cf. section 5.4.2.1).

2. It has a list of multiples names, called *aliases*. Aliases are used to designate *equivalent* electrical parameters exactly as in devices (cf. section 5.4.2.1).

6.3.2.2 Arc Definition

As in devices (cf. section 5.4.2.2), an *arc* represents a weighted dependence of one node v on another node u. Therefore, an arc is *directed* since it has a propagation direction from node u to node v. It is said that v depends on u. This is formally written as $v \xleftarrow{w_i} u$. It means also that $v = w_i \times u$.

6.3.2.3 Dependency Rule Definition

A *dependency rule* (or simply *dependency*) expresses electrical dependencies of a node v on other nodes $u_1, u_2, \cdots, u_n$. This is formally written as $v \xleftarrow{w_i} u_1, u_2, \cdots, u_n$. Multiple nodes can be computed from the rule simultaneously. This is expressed as $v_1, v_2, \cdots, v_m \xleftarrow{w_i} u_1, u_2, \cdots, u_n$. A dependency rule possesses some properties:

1. It has a rule type: *constraint*, *operator*, or *designer-defined procedure (DDP)*.
2. It has a constant weight. It has a default value of 1.0.
3. It has a list of affected nodes. These are the nodes computed by applying the rule.
4. It has a list of affecting nodes. These are the nodes that are used to compute an affected node.
5. It has a name of either an operator or a designer-defined procedure.

Dependency rules of constraint and operator types were illustrated for devices in section 5.4.2.3. New types of dependency rules are added to the *module dependency graph* definition:

1. **Extrinsic Device Constraints:** This type of constraints were illustrated in section 5.3.4. Their application depends on the context where the devices are used. They are not part of the device. They are imposed by the external environment using the device. For instance, a module level can impose this type of constraints on its instantiated devices. For a detailed example, please refer to section 5.3.4.
2. **Extrinsic Module Constraints:** This type of constraints depends on the context where the modules are used. They are imposed by the external environment using the module. For instance, a higher module level can impose this type of constraints on its instantiated lower-level modules. This is done by specifying equality constraints relating the input parameters of the lower level modules.
3. **Designer-Defined Procedures (DDPs):** In many occasions, the designer would like to code a portion of knowledge into a reusable procedure and execute it as part of the dependency graph. For this situation, a DDP is used to express the dependency of one output parameter on its input parameters. As opposed to an operator which executes a predefined procedure

in CAIRO+, a DDP is integrated into the dependency graph and called when its single output parameter needs to be computed.

Appendix D shows how CAIRO+ is used as a dependency language for modeling and design. The language constructs related to DDPs are further detailed in D.3.

6.4 Bottom-Up Construction of Module Dependency Graphs

In this section, the algorithms developed to construct module dependency graphs for analog circuits are presented. The construction of module dependency graphs is performed using a hierarchical bottom-up approach. The approach consists of hierarchically merging dependency graphs for children devices and lower-level modules. To go through the hierarchy, an important step is the identification of equipotentials. This will be explained first.

6.4.1 Identification of the Equipotentials

The first step towards generating design plans is the automatic identification of the equipotentials in the current module level. An equipotential is defined as the set of all interconnected interface terminals. Terminals can be *external* or *internal* to the current module level. External terminals are on the external interface of the current module level. Internal terminals are on the external interface of the children devices and lower-level modules instantiated in the current module level. To identify equipotentials, the netlist in the current module level is traversed. Then, interconnected terminals belonging to the same equipotential are preserved by adding their terminal names as aliases of the same equipotential graph node. Fig. 6.3 shows an example of a module consisting of device instance A, device instance B and module instance C. The possible equipotential nodes are (e_1,A/s), (e_2,A/g), (e_3,B/g), (e_4,B/s), (e_5,C/y) and (A/d,B/d,C/x),

Let us illustrate how the equipotential node (A/d,B/d,C/x) is created. First interconnected interface terminals: A/d, B/d and C/x are identified. Then, these interface terminals are merged to form one equipotential node. The merging is done by converting each terminal name to an alias name for the same equipotential node. This technique is illustrated in Fig. 6.4. Finally, an equipotential graph node with multiple aliases is created.

6.4.2 Generation of the Reference Transistor Dependencies

This section illustrates the algorithms designed to generate the dependencies for the reference transistor. First, a general algorithm is presented. Then, its different blocks are detailed in subsequent sections. The purpose of each block is to generate the dependency for a specific class of operators. In addition, each block identifies which operator version to apply based on the known parameters. It is assumed that a parameter is known if it is specified by the designer or known

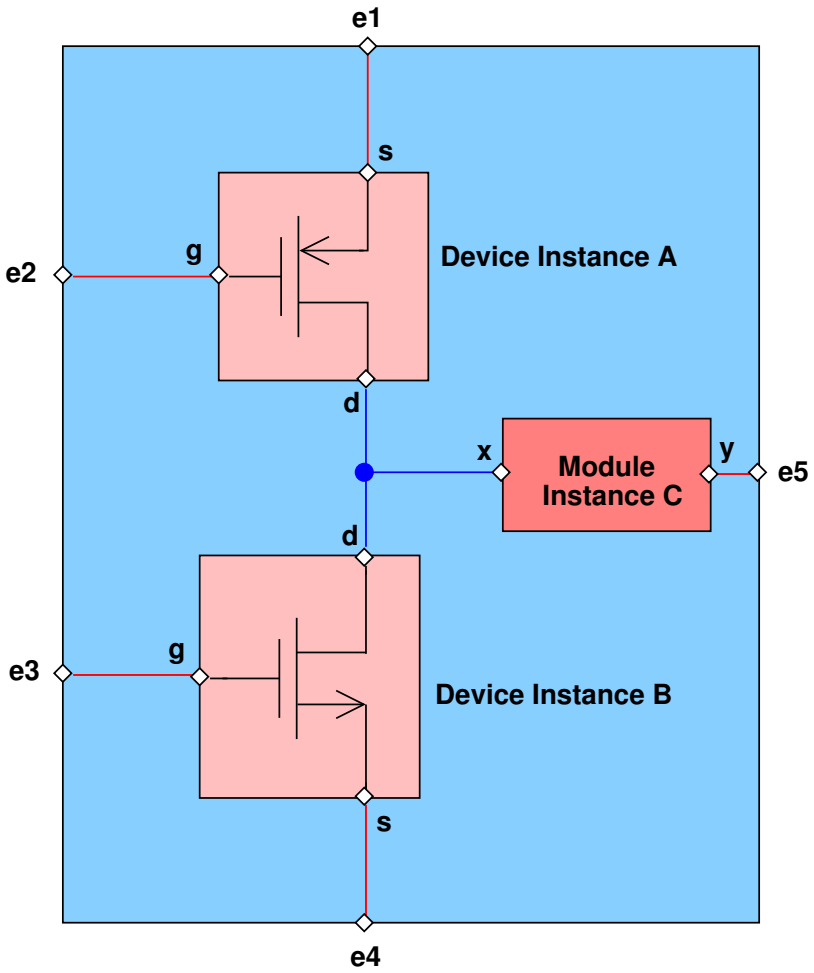

Figure 6.3: *Equipotential consisting of interconnected terminals.*

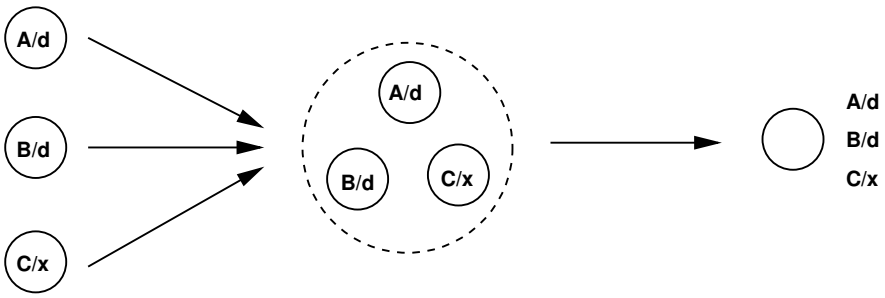

Figure 6.4: *Adding terminal names to the same equipotential node.*

as a result of a previously generated dependency. As will be explained, the generation of the dependencies is performed in the *designer mode* and the *simulator mode*.

6.4.2.1 General Algorithm

The flowchart of the main routine executed for the reference transistor of device is shown in Fig. 6.5. It performs the following:

1. The dependency of the *drain-source* current is generated.
2. The flowchart then determines whether the synthesis is performed in the designer mode or the simulator mode. In designer mode, either V_{eg} or V_{GS} are specified but not both. In the simulator-mode, W and L are specified.
3. The flowchart then continues by generating the source voltage dependency.
4. Then the interconnection configuration of the reference transistor is examined. If it is a diode-connected transistor, the gate/drain voltage dependency is generated. Otherwise, the gate voltage dependency is generated.
5. At the end, the width dependency is generated if it was not already generated.

6.4.2.2 Generation of Drain Current Dependency

This step of the general algorithm is further detailed in the routine flowchart shown in Fig. 6.6. The current I_{DS} is first examined. If it is specified, this step will end without any further action. Otherwise, the routine verifies V_{eg} against V_{GS}:

1. If both V_{eg} and V_{GS} are specified, the routine will display an error message stating that these two parameters cannot be specified simultaneously.
2. If neither V_{eg} nor V_{GS} are specified, the dependency takes both the gate voltage V_G and the source voltage V_S as affecting nodes.
3. If only V_{GS} is specified, the dependency takes either V_G or V_S as affecting node, whichever set. An error message is displayed if neither are specified.
4. If only V_{eg} is specified, the dependency takes both V_G and V_S as affecting nodes.

Note that every affecting node is first searched for. If it is previously created, i.e. as an equipotential node, it is retrieved and added to the dependency. Otherwise, it is created as a new graph node and added to the dependency as an affecting node. The routine proceeds as follows:

1. It searches or creates nodes $Temp$, L, V_D and W as affecting nodes.

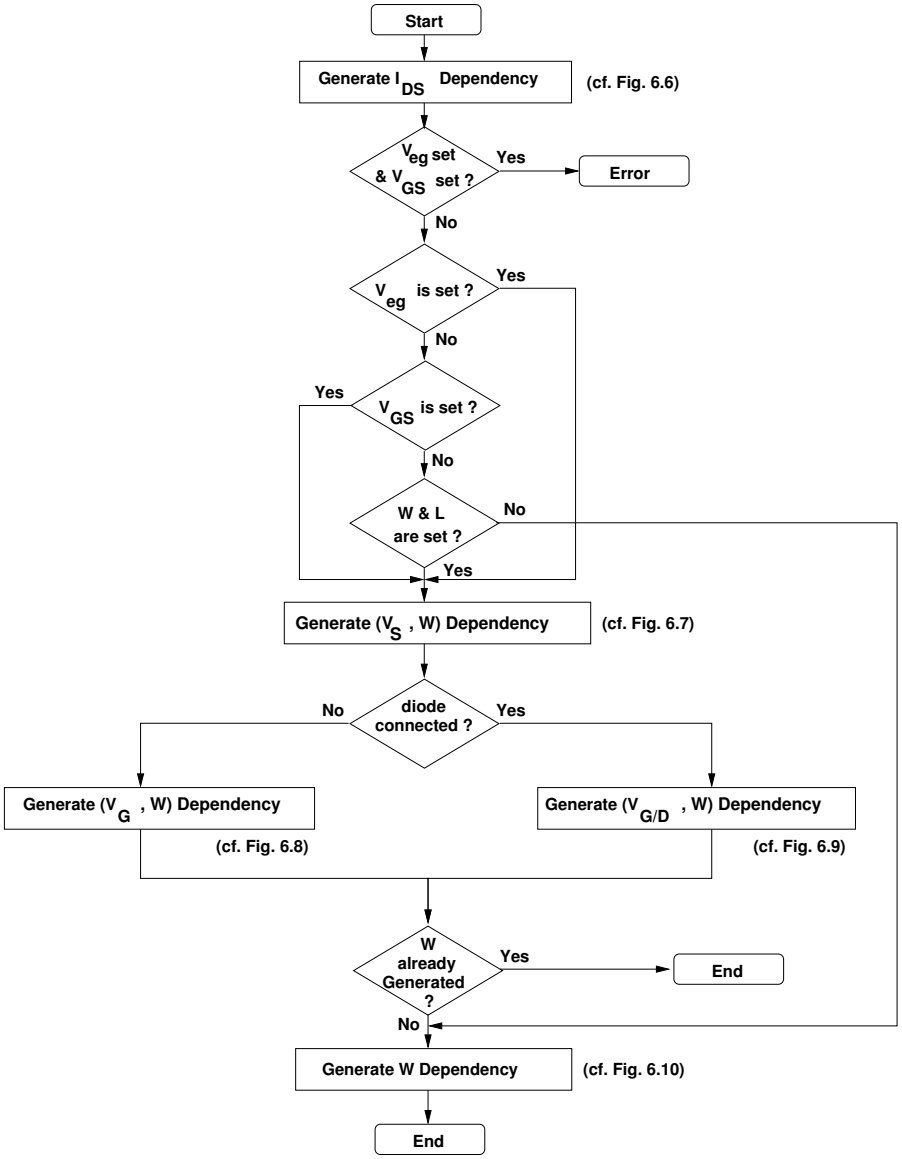

Figure 6.5: *Dependency generation for the reference transistor.*

2. Then, it searches or creates either V_{eg} or V_{GS}, if specified, as affecting node.

3. In addition, it searches or creates node V_B, if the reference transistor was not bulk-source connected.

4. The current I_{DS} is searched or created. It is then added as an affected node in the dependency.

5. The operator name is set and its version is determined based on the known affecting nodes.

6. The dependency is finally created and the graph node I_{DS} is marked as a known affected node if all its affecting nodes are known or specified.

Note that the drain-source current dependency is generated essentially in simulator mode since the current is computed in this mode.

6.4.2.3 Generation of Source Voltage Dependency (V_S + W)

This step of the general algorithm is further detailed in the flowchart shown in Fig. 6.7:

1. The source voltage V_S is first examined. If it is specified, this step will end without any further action.

2. Otherwise, the flowchart proceeds by searching or creating nodes $Temp$, I_{DS} and L as affecting nodes for both W and V_S dependencies.

3. Next, the flowchart examines V_{eg}. If it is specified, then its corresponding node is searched or created as an affecting node and added to both W and V_S dependencies.

4. The search and creation steps is also performed for V_{GS}, V_D, V_B and V_G.

5. The routine proceeds by checking if W is specified at that point. If it is the case, it is searched or created then added as an affecting node to only the V_S dependency. Otherwise, W is added as an affected node to the W dependency. This is followed by setting the operator name and version needed to compute W. Then, the dependency for W is then created.

6. In the same manner, the checking for V_S till the creation of the V_S dependency proceeds.

7. The flowchart ends by marking the affected nodes W and V_S as known nodes if their corresponding affecting nodes are known or specified.

Note that the source voltage dependency is generated in both designer mode and simulator mode.

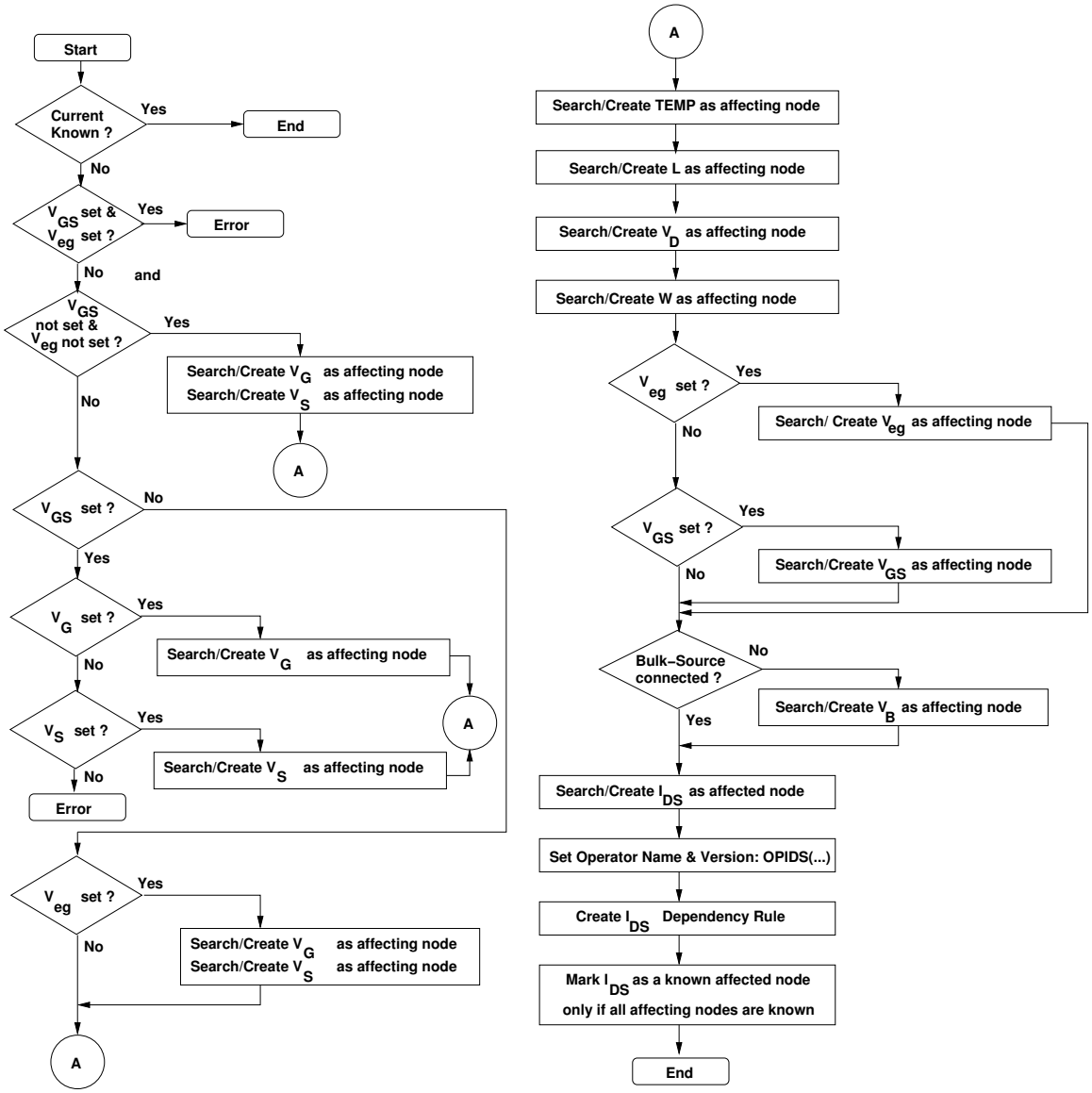

Figure 6.6: *Dependency Generation for the operator* $OPIDS(...)$.

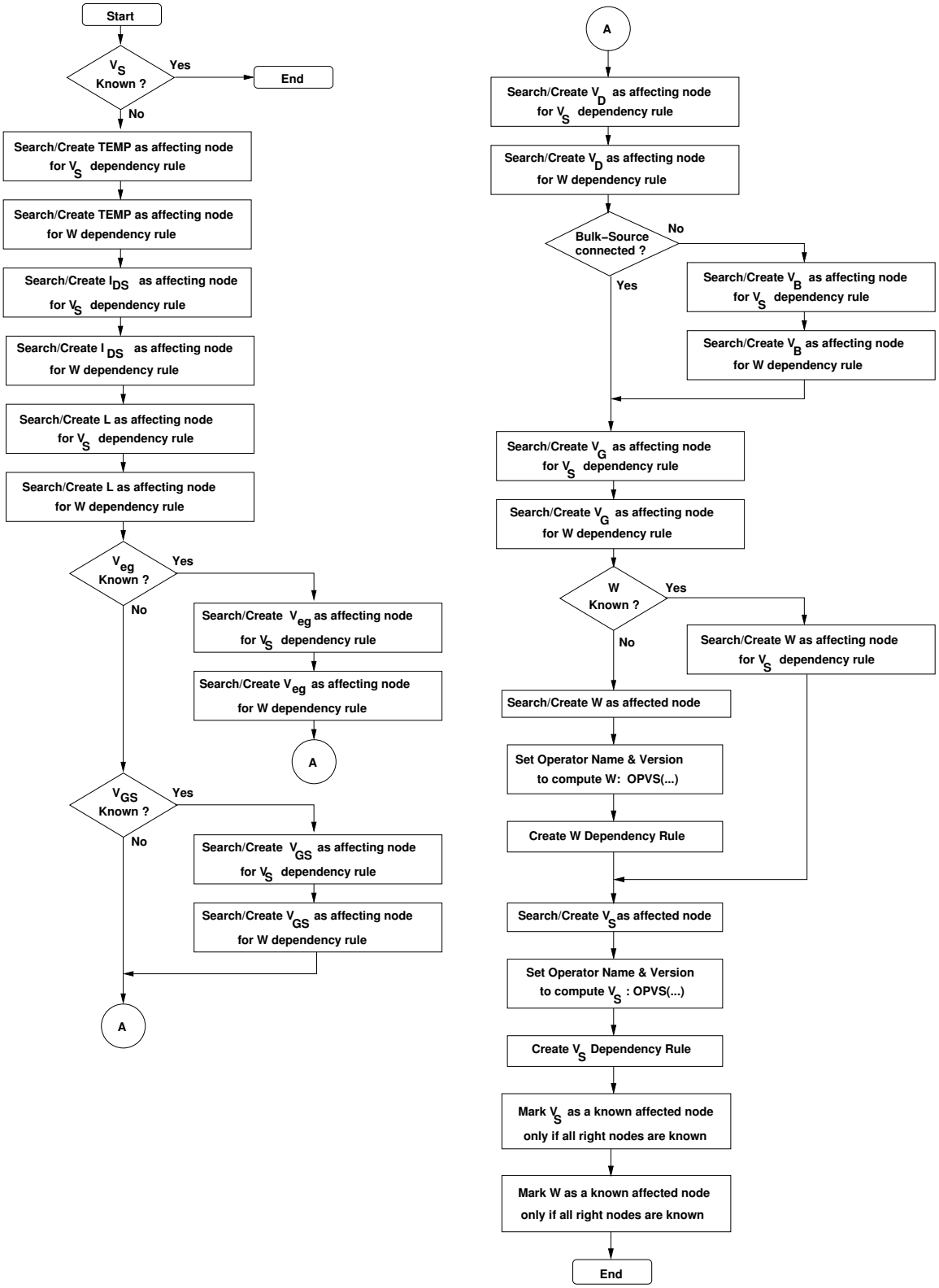

Figure 6.7: *Dependency Generation the operator* $OPVS(...)$.

6.4.2.4 Generation of Gate Voltage Dependency (V_G + W)

This step of the general algorithm is further detailed in the flowchart shown in Fig. 6.8:

1. The gate voltage V_G is first examined. If it is specified, this step will end without any further action.
2. Otherwise, the routine proceeds by searching or creating nodes $Temp$, I_{DS} and L as affecting nodes to both W and V_G dependencies.
3. Next, the routine examines V_{eg}. If it is specified, then its corresponding node is searched or created as an affecting node and added to both W and V_G dependencies.
4. The search and creation steps is also performed for V_{GS}, V_D, V_B and V_S.
5. The routine proceeds by checking if W is specified at that point. If it is specified, it is searched or created then added as an affecting node to only the V_G dependency. Otherwise, W is added as an affected node to the W dependency. This is followed by setting the operator name and version needed to compute W. Then, the dependency for W is then created.
6. In the same manner, the checking for V_G till the creation of the V_G dependency proceeds.
7. The routine ends by marking the affected nodes W and V_G as known nodes if their corresponding affecting nodes are known or specified.

Note that the gate voltage dependency is generated in both designer mode and simulator mode.

6.4.2.5 Generation of Gate/Drain Voltage Dependency ($V_{G/D}$ + W)

This step of the general algorithm is further detailed in the flowchart shown in Fig. 6.9:

1. The gate/drain voltage $V_{G/D}$ is first examined. If it is specified, this step will end without any further action.
2. Otherwise, the routine proceeds by searching or creating nodes $Temp$, I_{DS} and L as affecting nodes to both W and $V_{G/D}$ dependencies.
3. Next, the routine examines V_{eg}. If it is specified, then its corresponding node is searched or created as an affecting node and added to both W and $V_{G/D}$ dependencies.
4. The search and creation steps is also performed for V_{GS}, V_D, V_B and V_S.
5. The routine proceeds by checking if W is specified at that point. If it is specified, it is searched or created then added as an affecting node to only the V_G dependency. Otherwise, W is added as an affected node to the W dependency. This is followed by setting the operator name and version needed to compute W. Then, the dependency for W is then created.

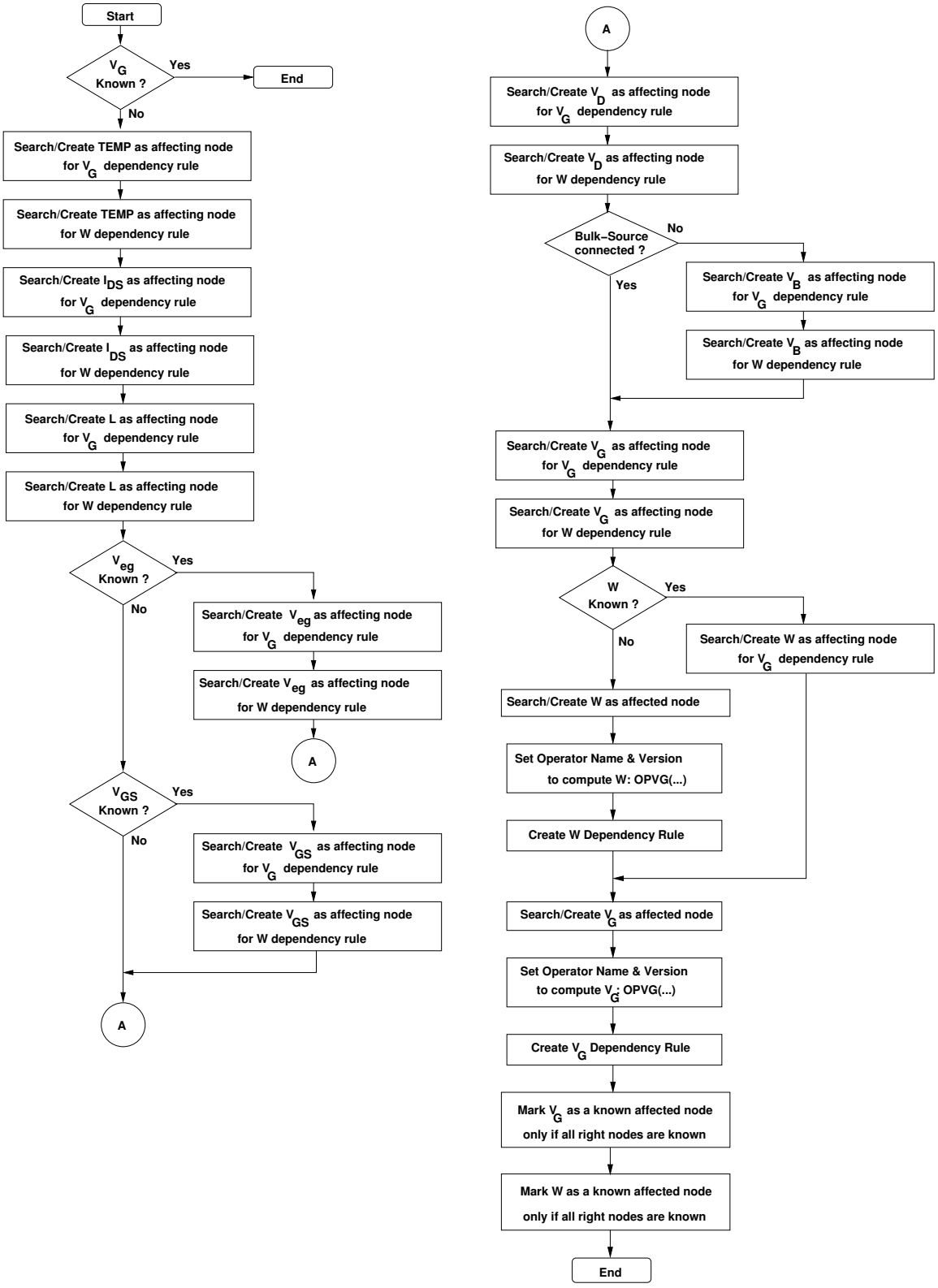

Figure 6.8: *Dependency Generation for the operator* $OPVG(...)$.

6. In the same manner, the checking for $V_{G/D}$ till the creation of the $V_{G/D}$ dependency proceeds.

7. The routine ends by marking the affected nodes W and $V_{G/D}$ as known nodes if their corresponding affecting nodes are known or specified.

Note that the gate/drain voltage dependency is generated in both designer mode and simulator mode.

6.4.2.6 Generation of Width Dependency

This step of the general algorithm is further detailed in the routine flowchart shown in Fig. 6.10. The width W is first examined. If it is specified, this step will end without any further action. Otherwise, the routine verify V_{eg} against V_{GS}:

1. If both V_{eg} and V_{GS} are specified, the routine will display an error message stating that these two parameters cannot be specified simultaneously.

2. If neither V_{eg} nor V_{GS} are specified, the dependency takes both the gate voltage V_G and the source voltage V_S as affecting nodes.

3. If only V_{GS} is specified, the dependencies takes either V_G or V_S as affecting node, whichever set. An error message is displayed if neither are specified.

4. If only V_{eg} is specified, the dependencies takes both V_G and V_S as affecting nodes.

Note that each affecting node is first searched for. If it is previously created, it is retrieved and added to the dependency. Otherwise, it is created as a new graph node and added to the dependency as an affecting node. The routine proceeds as follows:

1. It searches or creates nodes $Temp$, L, V_D and I_{DS} as affecting nodes.

2. Then, it searches or create either V_{eg} or V_{GS}, if specified, as affecting node.

3. In addition, it searches or creates V_B, if the reference transistor was not bulk-source connected.

4. The width W is searched or created. It is then added as an affected node in the dependency.

5. The operator name is set and its version is determined based on the known nodes.

6. The dependency is finally created and the graph node W is marked as a known affected node if all its affecting nodes are known or specified.

Note that the width dependency is essentially generated in designer mode to compute the width of transistors

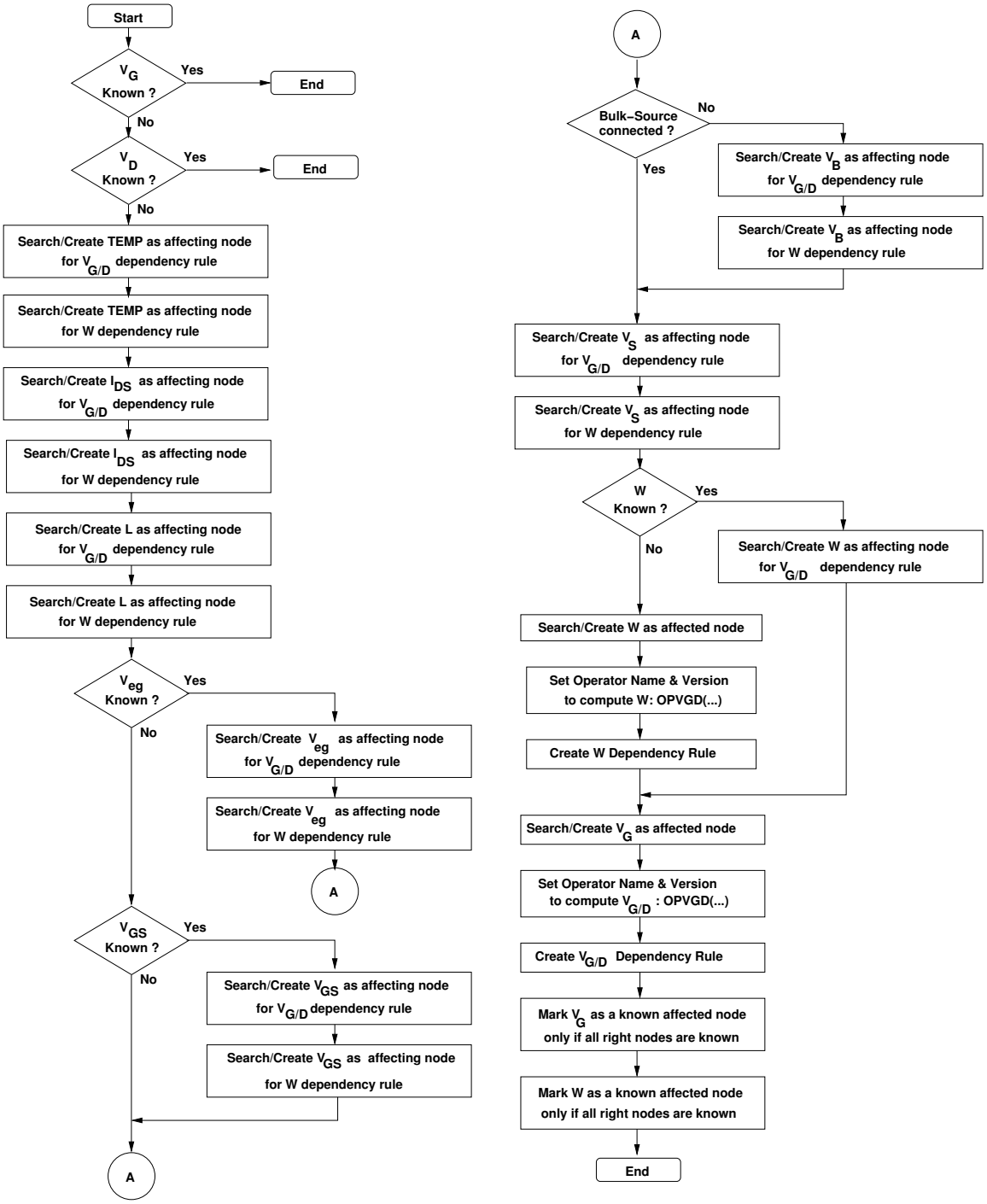

Figure 6.9: *Dependency Generation for the operator* $OPVGD(...)$.

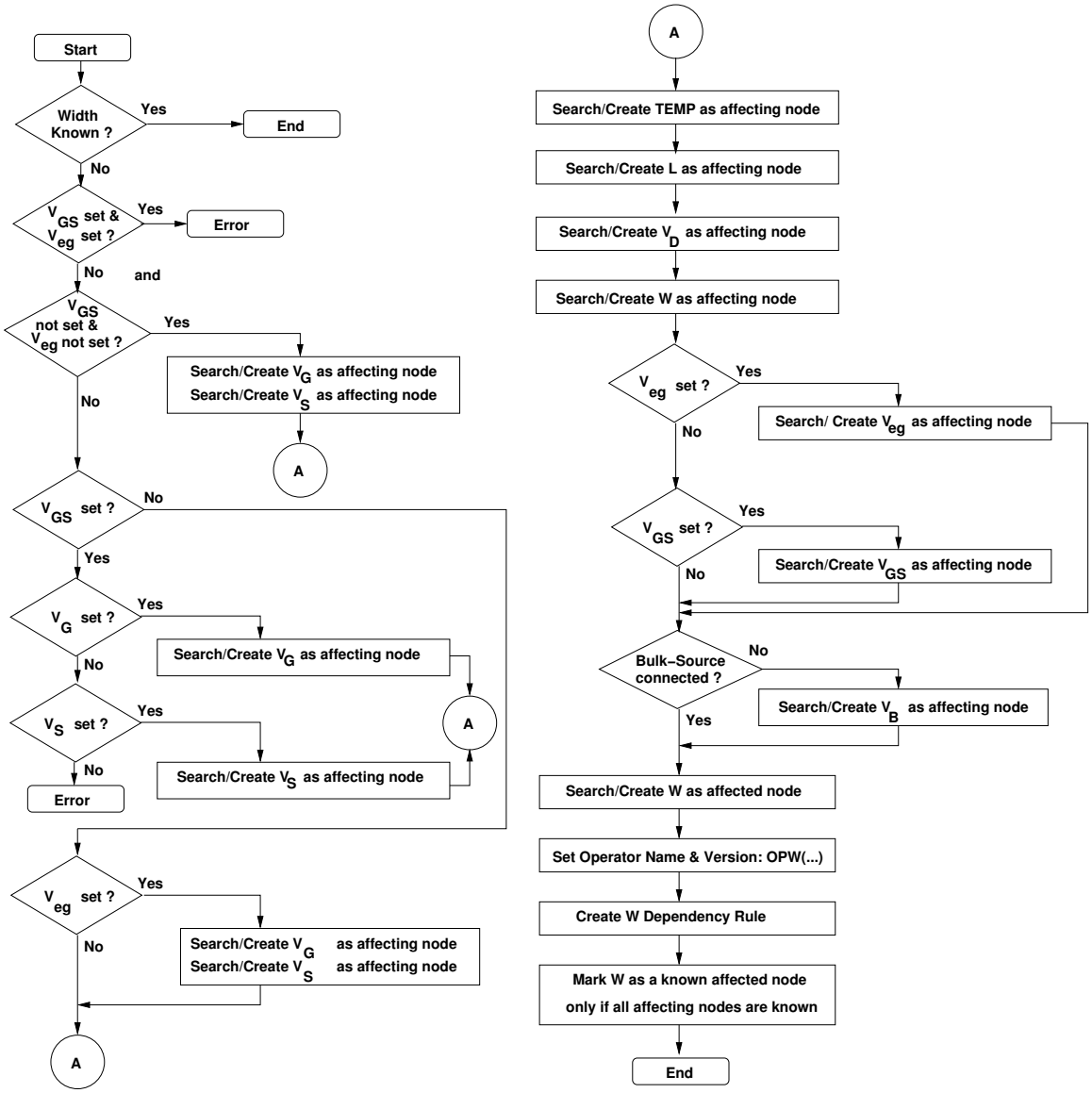

Figure 6.10: *Dependency Generation for the operator* $OPW(...)$.

6.4.3 Merging Dependencies of Children Devices and Lower-Level Modules

In the previous chapter, a methodology has been proposed to generate the design plans for devices. Once design plans exist for children devices and lower-level modules, the design plans for the current module level can be initially constructed by merging the design plans of children devices and lower-level modules. We call children devices and lower-level modules as *children generators*. Since a design plan is represented by a dependency graph, the merging is done on the dependency graph nodes and arcs. The merging starts by enumerating all children generators. For each child generator, the list of dependencies in its pool is enumerated. For each child dependency, the affected node is checked if it was already created in the pool of the current module level. If it exists, then its aliases in the pool are merged to its aliases in the child dependency and the node is made universal by merging it to all nodes in the pool having common aliases. Then, the node is set as the affected node of a newly created dependency in the pool of the current module level. The same technique is applied for every affecting node of the child dependency. A final step done on the newly created dependency, is to mark the affected node as known if all its affecting nodes are known at this stage.

6.4.4 Independence from Device Ordering

Since the design plan of a module is generated by merging the dependency graph of the children generators, it is important to ensure that the resulting design plan is independent from the order by which children generators have been merged. This simply means that for the same set of constraints and hypotheses set by the designer, the generated design plan of the circuit will always be the same regardless of the device ordering. To achieve the independence from device ordering, we identified three main problems that need to be solved:

6.4.4.1 Single Constraint/Single Operator Problem

Proposition 1 *A constraint always has higher preference over an incident operator.*

Proof Let us suppose that a node N_B is affected by both a constraint and an incident operator, as illustrated in Fig. 6.11(a). Since the constraint is a condition that is imposed by the designer and should be satisfied, it is a persistent knowledge. On the other hand, an incident operator is an information that has been generated to complete the knowledge based on designer's hypothesis. Since the knowledge is already ensured by the constraint, the incident operator could be safely removed. Then the remaining graph is corrected. This results in the graph shown in Fig. 6.11(b). Therefore, constraints should have higher preference over an incident operator as stated by proposition 1.

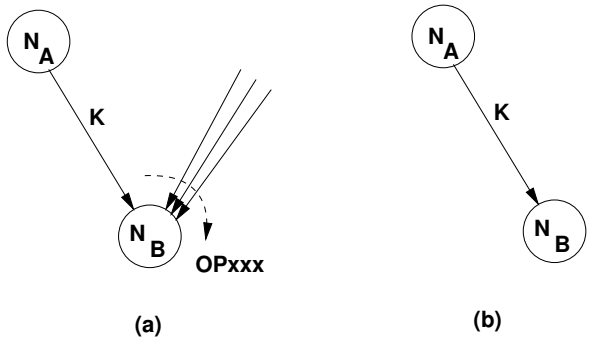

Figure 6.11: *Preference of a constraint over an incident operator: (a) Conflict, (b) Resolution*.

6.4.4.2 Single Constraint/Multiple Operator Problem

Definition A directed cycle is called *first-order directed cycle* if it is a directed cycle between only two operators. In this case, each operator depends on a parameter computed from the other operator.

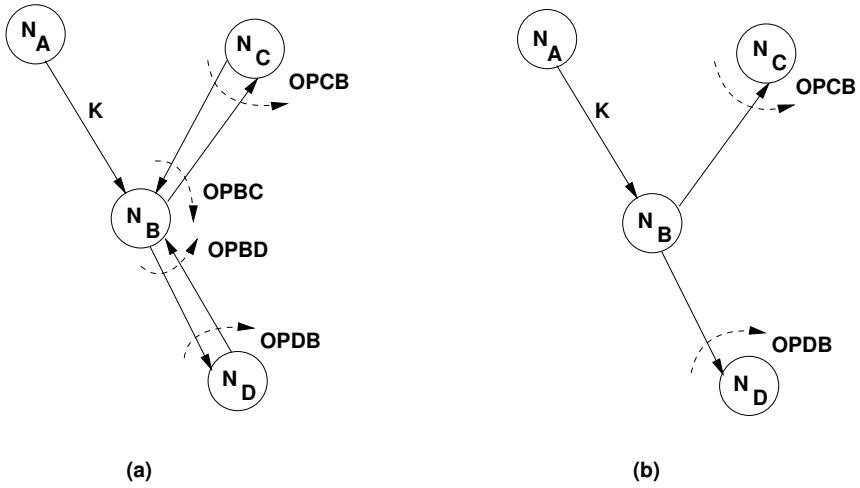

Figure 6.12: *Preference of a constraint over multiple incident operators: (a) Conflict, (b) Resolution*.

Fig. 6.12(a) shows examples of first-order directed cycles: (N_B,N_C) and (N_B,N_D). Node N_B is the common node between the two first-order directed cycles. Since a constraint always has a higher preference over each incident operator, one can apply proposition 1 to resolve this situation in the figure. After resolution, the resulting graph will evolve in only one direction as shown in Fig. 6.12(b). Proposition 2 states that the previous proposition can be applied in this case.

Proposition 2 *Proposition 1 can be applied for the case of having multiple incident operators having first-order directed cycles and a single constraint, all affecting the same node.*

Proof Since the constraint has a higher preference over each incident operator, then proposition 1 applies for every operator.

6.4.4.3 No Constraint/Multiple Operator Problem

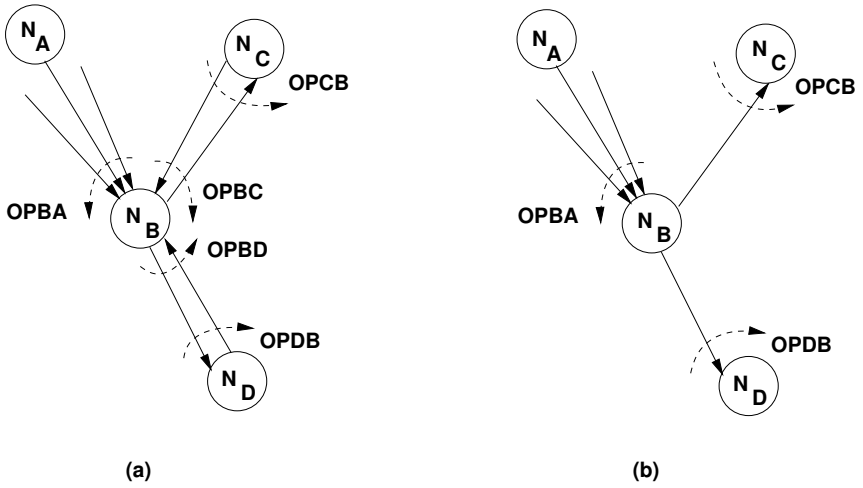

Figure 6.13: *Multiple incident operators: (a) Conflict, (b) Resolution*.

Fig. 6.13(a) shows examples of first-order directed cycles: (N_B,N_C) and (N_B,N_D). Node N_B is the common node between the two first-order directed cycles. In this case, no constraint exist. But there exists an operator that has no cycle directly affecting the common node N_B. We now present proposition 3 that will resolve the situation as shown in Fig. 6.13(b).

Proposition 3 *In the case of multiple incident operators that affect a common node: if no additional constraint exists, only one operator needs to be noncyclic in order to remove first-order directed cycles at that node.*

Proof In a first-order directed cycle, each operator has a missing parameter that is computed by the other operator. Since operators in a first-order directed cycle are generated to complete the knowledge and are not considered persistent, one can eliminate them in order to respect a noncyclic persistent operator.

6.5 Dealing with Different Aspects in Analog Design

6.5.1 Dealing with Under-Specified Designs

An *under-specified* design is a design which does not have sufficient parameters (or degrees of freedom) to be specified. For example , suppose that a design has two parameters u and v that

depend on each other as depicted in Fig. 6.14(a). The parameter u cannot be evaluated without knowing v and vice versa. This means that we do not have sufficient degrees of freedom to specify the problem. In order to solve the problem, one could choose u as a degree of freedom and then compute v as shown in Fig. 6.14(b). Or, choose v as another possible degree of freedom and then compute u as shown in Fig. 6.14(c). We conclude that insufficient degrees of freedom is characterized by the formation of directed cycles.

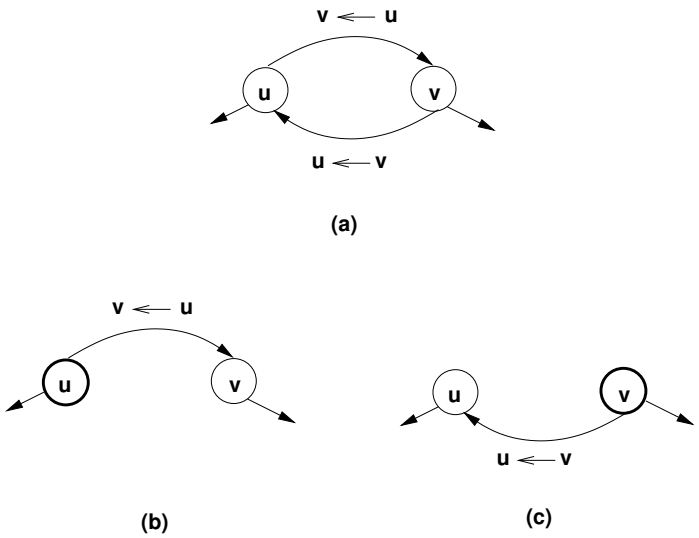

Figure 6.14: *Under-specified design dependency.*

In general, if many degrees of freedom are missing, directed cycles will contain more than two parameters nodes. The general rule is to detect if dependency graphs contains directed cycles and to choose at least one parameter to specify among the parameters forming the directed cycle. Fig. 6.15(a) shows many parameters depending on each other. To resolve the dependencies, y has been chosen as degree of freedom as shown in Fig. 6.15(b).

To detect directed cycles in the dependency graphs, the algorithm [Tiernan70], previously developed at IBM, has been implemented. Once detected, directed cycles are displayed for the designer to inspect them and choose one favorable degree of freedom to solve each of them. Another possibility is to choose, by default, the first parameter of a directed cycle as the additional degree of freedom. Once selected, the designer has to set this parameter in the corresponding module or device level to transform it to a controllable degree of freedom.

6.5.2 Dealing with Over-Specified Designs

An *over-specified* design is a design which has degrees of freedom more than actually required to describe its physical dependencies. We show that dependency graphs allow us to detect and resolve over-specified designs. The study of those designs gives lots of insight into the circuit design

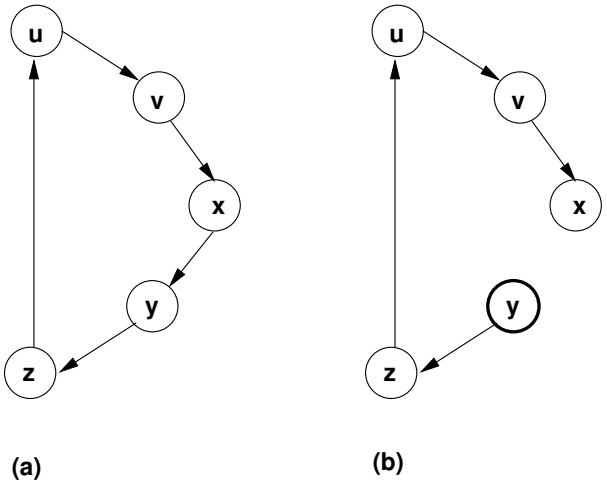

Figure 6.15: *Directed cycles consisting of many parameters*.

issues. We show that the design representation itself becomes an efficient aid in understanding and resolving these issues.

Since a design plan represents a consistent knowledge about the circuit, it should not contain any inconsistency. Inconsistency may appears as conflicting hypothesis [Wu94]. Mainly, we investigate the problem of systematic offset that appears in the design of amplifiers. We prove that a systematic offset appears as a conflicting hypothesis in the design knowledge. Its location is determined and later used to evaluate it precisely. The whole method is fully automated and integrated inside the CAIRO+ framework. It does not require any designer intervention.

6.5.2.1 Systematic Offset Voltage

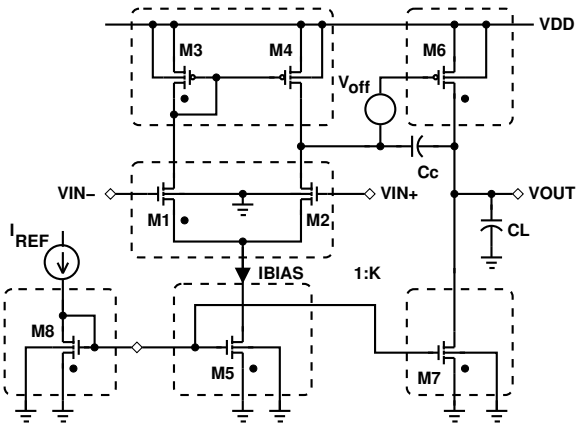

Figure 6.16: *Single-Ended Two-Stage Amplifier*.

The problem of offset is encountered during the design of analog circuits such as operational amplifiers, comparators, A/D, D/A, ... If both inputs of an ideal op-amp are connected to the same common-mode input potential, the output potential is equal to zero. This is not the case in real circuits due to *systematic* and *random* offsets. Systematic offset depends on the circuit design. Random offset comes from random fluctuations of physical and technological parameters along the chip. In order to bring the output to zero, it is therefore required to apply a proper input offset potential at the input terminals.

Let us examine the two-stage amplifier in Fig. 6.16. For a given capacitance load C_L and a phase margin PM, increasing $g_{m,M6}$ lowers the value of the compensation capacitance C_C, hence, increases the gain bandwidth product $GBW \approx \frac{g_{m,M1}}{C_C}$. In order to increase $g_{m,M6}$ in strong inversion, where:

$$g_{m,M6} \approx \frac{2I_{M6}}{V_{GS,M6} - V_{th,M6}} \approx \frac{2I_{M6}}{V_{eg,M6}} \tag{6.1}$$

the overdrive voltage $V_{eg,M6} = V_{GS,M6} - V_{th,M6}$ should be lowered. This requirement conflicts with the arbitrary potential $V_{DS,M4}$. This conflict imbalances the amplifier. In order to balance the amplifier, a degree of freedom is created by liberating $V_{D,M4}$. The virtual difference between $V_{D,M4}$ and $V_{G,M6}$ is the systematic offset voltage appearing at the output of the first stage. To bring this offset to the input of the amplifier, we divide it by the gain of the first stage amplifier,

$$V_{i,off} \approx (V_{D,M4} - V_{G,M6}) \cdot \frac{g_{ds,M2} + g_{ds,M4}}{g_{m,M1}} \tag{6.2}$$

Generalizing this principle, any conflict in a node potential can be solved by inserting a systematic offset voltage.

6.5.2.2 Conflict Detection

Table 6.1: *Sizing & biasing operators for the amplifier in Fig. 6.16.*

Operator	Definition
OPVS(V_{eg}, V_B)	$(V_S, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_G, V_B$
OPVG(V_{eg})	$(V_G, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_D, V_S$
OPVGD(V_{eg})	$(V_G, V_D, V_B, V_{th}, W) \Leftarrow Temp, I_{DS}, L, V_{eg}, V_S$
OPW(V_G, V_S)	$(W, V_B, V_{th}) \Leftarrow Temp, I_{DS}, L, V_D, V_G, V_S$
OPIDS(V_G, V_S)	$(I_{DS}, V_B, V_{th}) \Leftarrow Temp, W, L, V_D, V_G, V_S$

During construction of the module dependency graph, conflicts appear as multiple operators which are computing the same unknown parameter, such as node voltage. Actually, there is no guarantee that those operators will calculate equal node voltages. This situation creates a conflict

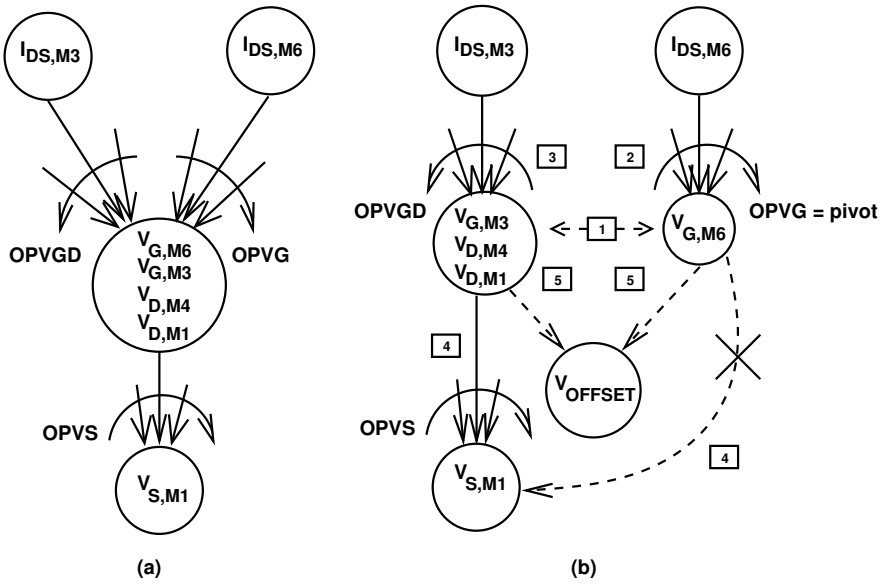

Figure 6.17: *Conflicts between operators: (a) Detection, (b) Resolution. The resolution steps are enumerated in sequence. Nodes represent parameters, solid arcs represent dependency between parameters, labelled arcs are operators, and dotted arcs are either added or removed dependencies.* .

as illustrated in Fig. 6.17(a). First, let us assume that the current mirror (M_3,M_4) is ideal, i.e. $V_{G,M3} = V_{D,M4}$. In Fig. 6.17(a), $V_{G,M3}$ and $V_{D,M4}$ share the same node since they should respect this equality constraint. Operator *OPVGD*, listed in Table 6.1, is used to compute $V_{G,M3}$ and $V_{D,M4}$ from the known quantities of the current mirror, e.g. $I_{DS,M3}$. Also, operator *OPVG* in the same table is used to compute $V_{G,M6}$ from known quantities of M_6, e.g. $I_{DS,M6}$. Unfortunately, $V_{D,M4}$ and $V_{G,M6}$ form the same equipotential. Therefore $V_{G,M6}$ share the same node with $V_{G,M3}$, $V_{D,M4}$ and $V_{D,M1}$. Since, both operators *OPVGD* and *OPVG* compute the same node, they are conflicting by definition. Hence, one degree of freedom is needed to resolve this conflict. This degree of freedom should be inserted in the graph to transform it to a conflict-free one.

6.5.2.3 Conflict Resolution

In order to resolve conflicts between operators, we propose the technique of *node splitting*. In this technique, a *pivot* operator is selected. The pivot operator is defined as an operator computing either the gate voltage (i.e. *OPVG*) or the source voltage (i.e. *OPVS*) of a MOS transistor. Both operators are defined in Table 6.1. It was shown in Fig. 4.1 that the drain potential may be fixed or determined from its connection to either a gate or a source terminal of another transistor. Therefore, it is assumed that offsets are attached to only source or gate terminals of a MOS transistor. Diode-connected transistors are excluded from this definition.

Once the pivot operator is selected, the parameters that do not match are separated into a

split node. Each of the conflicting operators are corrected to point to the appropriate node. Then all graph dependencies are repaired with respect to the original and split nodes. This is done using instance pathname equivalence between dependencies and parameters. Once repaired, a systematic offset node is created in the graph. Its dependencies on the original and split nodes are registered. It is inserted as one degree of freedom in the dependency graph to transform it into a conflict-free one. This way, knowledge in the graph becomes consistent. In Fig. 6.17(b), the pivot operator is *OPVG* since it computes only a gate voltage. After splitting in step 1, the original node has only $V_{G,M6}$ and the split node has $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. Since *OPVG* was originally created to compute $V_{G,M6}$, it is corrected in step 2 to point to the original node $V_{G,M6}$. *OPVGD* is corrected in step 3 to point to the split node $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. In step 4, the dependency of $V_{S,M1}$ on the original node is corrected to point to the split node. This is performed using equivalence on instance pathname M_1. Since $V_{S,M1}$ depends logically on parameters of M_1, it is not correct to have it depending on the original node $V_{G,M6}$ that does not contain any parameters of M_1. Therefore, the dependency of $V_{S,M1}$ on original node $V_{G,M6}$ is replaced by a dependency on the split node $\{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$ which contains $V_{D,M1}$. In step 5, the offset node is added to the graph. It depends on both the original and the split nodes. It is defined as $V_{OFFSET} = V_{G,M6} - \{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. The graph is now conflict-free.

6.5.2.4 Computing Systematic Input Offset in Designer Mode

In the designer mode, the systematic offset is computed from the dependency graph of the amplifier as $V_{OFFSET} = V_{G,M6} - \{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. To bring it to the input of the amplifier, we divide it by the static gain of the first stage as given by equation 6.2 which is repeated here for convenience,

$$V_{i,off} \approx (V_{D,M4} - V_{G,M6}) \cdot \frac{g_{ds,M2} + g_{ds,M4}}{g_{m,M1}} \tag{6.3}$$

The more the equation of the static gain is precise, the better is the estimate of the systematic input offset.

6.5.2.5 Computing Systematic Input Offset in Simulator Mode

In simulator mode, the systematic offset is computed from the dependency graph of the amplifier as $V_{OFFSET} = V_{G,M6} - \{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$. To bring it to the input terminal, we reformulate problem as follows: *What is the potential of the amplifier positive input that will bring* $V_{OFFSET} = V_{G,M6} - \{V_{G,M3}, V_{D,M4}, V_{D,M1}\}$ *to zero.* In general, this is reformulated as the constraint equation 6.4 on offset. The equation solves for the value of the amplifier positive input parameter V_{INP} that will bring V_{OFFSET} to zero.

$$F_{offset}(V_{INP}) = V_{G,M6}(V_{INP}) - V_{D,M4}(V_{INP}) = 0 \tag{6.4}$$

We solve this equation using a graph-based Newton-Raphson algorithm that will be described in section 6.5.4.

6.5.3 Dealing with Negative Feedback Circuits

The circuit sizing and biasing method, presented in this chapter, is extended to deal with negative feedback circuits [Sedra91]. Negative feedback is applied to effect one or more of the following properties:

1. **Desensitize the gain:** that is, make the value of the gain less sensitive to variations in the value of circuit components, such as variations that might be caused by changes in temperature.
2. **Reduce nonlinear distortion:** that is, make the output proportional to the input. In other words, make the gain constant independent of signal level.
3. **Reduce the effect of noise:** that is minimize the contribution to the output of unwanted electric signals generated by the circuit components and extraneous interference.
4. **Control the input and output impedances:** that is, raise or lower input and output impedances by the selection of appropriate feedback topology.
5. **Extend the bandwidth** of an amplifier.

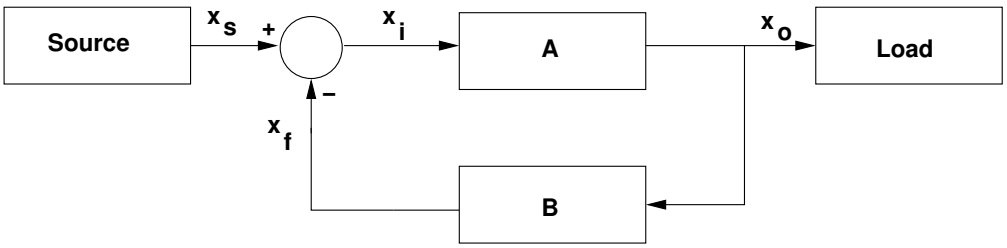

Figure 6.18: *Block diagram of a feedback circuit*.

Due to its numerous advantages, feedback circuits had to be analyzed by our proposed method. An example for a feedback circuit is shown in Fig. 6.18. The figure shows the basic structure of a feedback amplifier. Each arrow represents a voltage or a current signal. The *open-loop* amplifier has a gain A; thus its output x_o is related to the input x_i by

$$x_o = A \cdot x_i \tag{6.5}$$

The output is fed to the load as well as the feedback network, which produces a feedback signal x_f from the output. This feedback signal x_f is related to x_o by the *feedback factor* B,

$$x_f = B \cdot x_o \tag{6.6}$$

The feedback signal is subtracted from the source signal x_s, which is the input to the complete feedback amplifier, to produce the signal x_i, which is the input to the basic amplifier,

$$x_i = x_s - x_f \tag{6.7}$$

This subtraction makes the negative feedback. In essence, negative feedback reduces the signal that appears at the input of the basic amplifier. Note that in real circuits, the source, the load and the feedback network load the basic amplifier. That is the gain A depends on any of these three networks. This loading effect has to be taken into account during the evaluation of negative feedback circuits.

Note also that the gain of the feedback amplifier can be obtained by combining equations 6.5 through 6.7:

$$A_f = \frac{x_o}{x_s} = \frac{A}{1 + A \cdot B} \tag{6.8}$$

where $A \cdot B$ is called the loop gain. In the case of negative feedback, the loop gain $A \cdot B$ should be positive; that is, the feedback signal x_f should have the same sign as x_s thus resulting in a smaller difference signal x_i. Equation 6.8 indicates that for positive $A \cdot B$, the gain with feedback will be smaller than the open loop gain A by the quantity $1 + A \cdot B$, which is called the *amount of feedback*. For more information on feedback circuits, refer to [Sedra91].

Since a design plan should be represented by a directed acyclic graph (DAG), it should not contain any directed cycles. On the other hand, a negative feedback appears as a directed cycle in the dependency graph. This directed cycle is part of the structure and function of the circuit and cannot be eliminated. Otherwise, the circuit will not be functioning properly. Therefore, negative feedback is dealt with differently. In subsequent sections, we propose methods to compute negative feedback circuits and to represent them using dependency graphs, in both designer mode and simulator mode.

6.5.3.1 Negative Feedback Circuits in Designer Mode

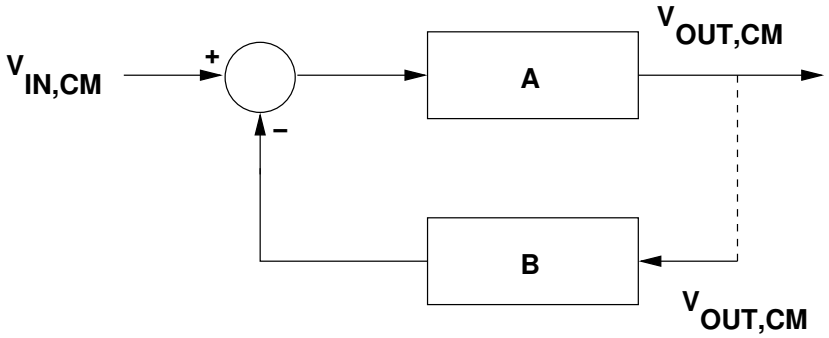

Figure 6.19: *Single negative feedback in designer mode.*

The designer mode is considered as an abstract view of the design. Under negative feedback the operating point evolves till achieving a steady state. This evolution cannot be simulated since the designer mode does not possess a general circuit DC solver. Therefore, the designer should impose approximate steady state conditions and deduces dimensions for the circuit.

As explained before, the negative feedback represents a directed cycle that has to be broken in order to obtain directed acyclic graphs. Therefore, under steady state conditions, the common-mode input to block B in Fig. 6.19 is equal to the common-mode output $V_{OUT,CM}$, producing zero differential input signal to the amplifier. Only under these circumstances, the negative feedback have no effect and can be broken. Normally, the common-mode voltage levels are known a priori, which facilitates this step. In this case, input and output common-mode levels become among degrees of freedom for the design. Once performed, the circuit is successfully sized and biased for steady state conditions.

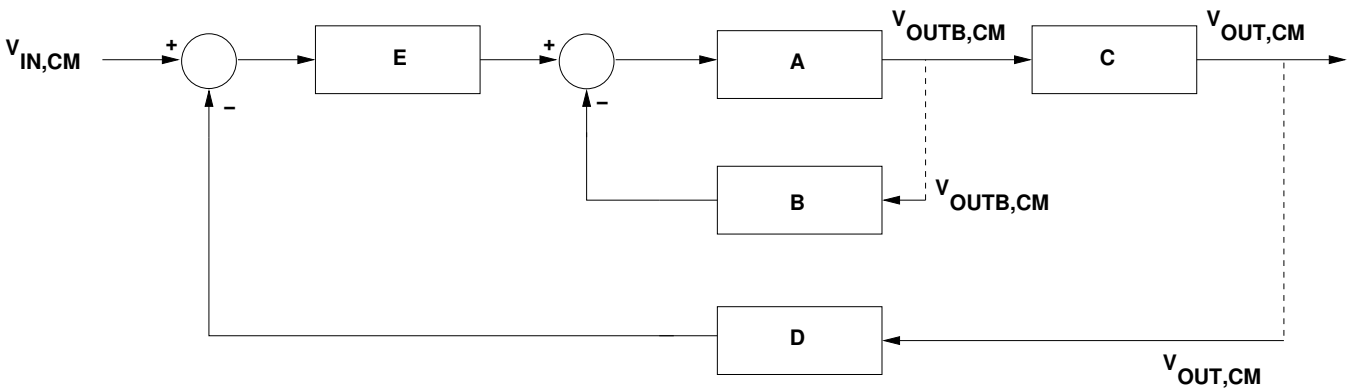

Figure 6.20: *Multiple negative feedbacks in designer mode*.

The above method is generalized for the case of multiple negative feedbacks that form multiple directed cycles in the circuit dependency graph. Those can be broken as shown in Fig. 6.20. The different common-mode levels have to be fixed for steady-state conditions and the circuit is then sized and biased.

6.5.3.2 Negative Feedback Circuits in Simulator Mode

As opposed to the designer mode, the simulator mode is considered as the accurate view which computes how the actual simulation will behave. In the simulator mode, the circuit is not hierarchical but flattened. In the flattened view, dimensions are given to all the transistors. Then, the circuit is simulated by computing branch currents and node voltages. Again, negative feedback poses problem as it adds directed cycles to the simulated dependency graph. Since the negative feedback is essential for proper functioning, it is not removed from the graph but represented

differently. From Fig. 6.21, we compute X' as follows

$$V_{OUT} = A \cdot (V_{IN} - X) \tag{6.9}$$
$$X' = B \cdot V_{OUT} \tag{6.10}$$

We substitute equation 6.9 into equation 6.10 to obtain X',

$$X' = A \cdot B \cdot (V_{IN} - X) = F_{feedback}(X) \tag{6.11}$$

Since $X' = X$ for the feedback to be effective, we can solve the equation $X' - X = 0$ to get the value of X that is the negative feedback signal. If $X' = F_{feedback}(X)$, then one solves

$$F_{feedback}(X) - X = 0 \tag{6.12}$$

where X is a selected parameter that represents the negative feedback signal and is used to solve for the feedback condition.

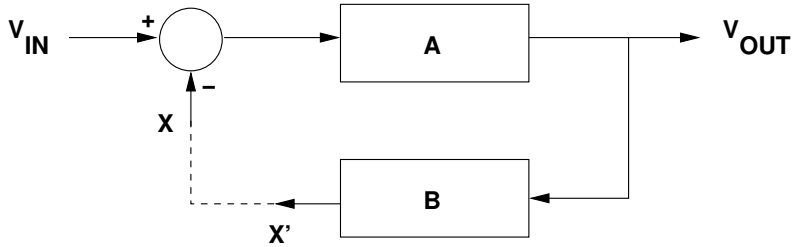

Figure 6.21: *Single negative feedback in simulator mode.*

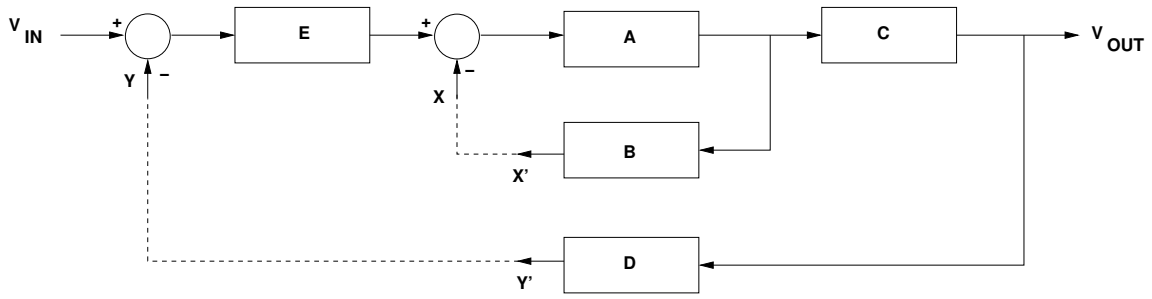

Figure 6.22: *Multiple negative feedbacks in simulator mode.*

For the case of multiple feedback signals in the block diagram in Fig 6.22, a negative feedback signal is selected for each negative feedback loop and is solved for each using equation 6.12.

Therefore, we deduce that

$$\begin{aligned} X' - X &= F_{1,feedback}(X) - X && (6.13)\\ &= A \cdot B \cdot [E \cdot (V_{IN} - Y) - X] - X = 0 && (6.14)\\ &= A \cdot B \cdot E \cdot (V_{IN} - Y) - (1 + A \cdot B) \cdot X = 0 && (6.15) \end{aligned}$$

$$\begin{aligned} Y' - Y &= F_{2,feedback}(Y) - Y = 0 && (6.16)\\ &= D \cdot C \cdot A \cdot [E \cdot (V_{IN} - Y) - X] - Y = 0 && (6.17)\\ &= D \cdot C \cdot A \cdot E \cdot V_{IN} - (1 + D \cdot C \cdot A \cdot E) \cdot Y - D \cdot C \cdot A \cdot X = 0 && (6.18) \end{aligned}$$

These equations are then solved together in X and Y using a graph-based Newton-Raphson algorithm that will be described in the next section.

6.5.4 Introducing a Unified Formulation for Simulator Mode

Before proposing a unified formulation for the simulator mode, we highlight another type of constraints that needs to be taken into account. This is the Kirchhoff's Current Law which states that the sum of currents entering a node is equal to the sum of currents flowing out of the node. This is mathematically expressed as

$$\sum_{i=1}^{m} I_{i,in} = \sum_{j=1}^{l} I_{j,out} \tag{6.19}$$

Equation 6.19 can be rewritten as an equality constraint as in equation 6.20. We call this type of constraint *Kirchhoff's current law constraint*. Note that in the last equation, currents are expressed in terms of a variable X which is considered as the degree of freedom that is solved for, in order to equalize the equation to zero.

$$F_{KCL}(X) = \sum_{i=1}^{m} I_{i,in}(X) - \sum_{j=1}^{k} I_{j,out}(X) = 0 \tag{6.20}$$

We represent now the unified formulation for the simulator mode. We would like to solve n nonlinear equations in n unknowns:

$$F_{KCL}(x_1, x_2, \cdots, x_n) = 0 \tag{6.21}$$

$$F_{offset}(x_1, x_2, \cdots, x_n) = 0 \tag{6.22}$$

$$F_{feedback}(x_1, x_2, \cdots, x_n) = 0 \tag{6.23}$$

or we have the vector of nonlinear equality constraints $\mathbf{F}(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \mathbf{F}_{\text{KCL}}(\mathbf{x}) \\ \mathbf{F}_{\text{offset}}(\mathbf{x}) \\ \mathbf{F}_{\text{feedback}}(\mathbf{x}) \end{bmatrix} = 0 \tag{6.24}$$

We call $\mathbf{F}(\mathbf{x})$ also the vector of *Newton-Raphson constraints*. Using the *Damped Newton-Raphson* algorithm [Coughran83], we have:

$$\mathbf{J}(\mathbf{x}_k) \cdot \mathbf{\Delta x}_k = -\mathbf{F}(\mathbf{x}_k) \tag{6.25}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma \cdot \mathbf{\Delta x}_k \tag{6.26}$$

where γ is the damping coefficient and $\mathbf{J}(\mathbf{x}_k)$ is the $n \times n$ jacobian matrix of $\mathbf{F}(\mathbf{x})$ with respect to $\mathbf{x}$. The jacobian matrix is defined as:

$$\mathbf{J}(\mathbf{x}_k) = \begin{bmatrix} \frac{\partial \mathbf{F}_{\mathrm{KCL}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^1} & \frac{\partial \mathbf{F}_{\mathrm{KCL}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^2} & \cdots & \frac{\partial \mathbf{F}_{\mathrm{KCL}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^\mathrm{n}} \\ \frac{\partial \mathbf{F}_{\mathrm{offset}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^1} & \frac{\partial \mathbf{F}_{\mathrm{offset}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^2} & \cdots & \frac{\partial \mathbf{F}_{\mathrm{offset}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^\mathrm{n}} \\ \frac{\partial \mathbf{F}_{\mathrm{feedback}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^1} & \frac{\partial \mathbf{F}_{\mathrm{feedback}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^2} & \cdots & \frac{\partial \mathbf{F}_{\mathrm{feedback}}(\mathbf{x}_\mathrm{k})}{\partial \mathrm{x}_\mathrm{k}^\mathrm{n}} \end{bmatrix} \tag{6.27}$$

Each element in the jacobian matrix $\mathbf{J}(\mathbf{x}_k)$ is defined as

$$\frac{\partial \mathbf{F}(\cdots, x_k^i, \cdots)}{\partial x_k^i} = \lim_{h_i \to 0} \frac{F(\cdots, x_k^i, \cdots) - F(\cdots, x_k^i - h_i, \cdots)}{h_i} \tag{6.28}$$

where h_i is an infinitely small step of computation.

To apply the above formulation, the designer states the types of constraints in $\mathbf{F}(\mathbf{x})$ required to describe the circuit behavior inside the module. Then, one degree of freedom is specified for each constraint. Finally, the system of nonlinear constraints is solved using equations 6.25 and 6.26.

Note the reduction in the size of $\mathbf{F}(\mathbf{x})$ since only the relevant node voltages are specified rather that all node voltages as in standard DC analysis.

Another important point is how to compute $F(\cdots, x_k^i, \cdots)$ and $F(\cdots, x_k^i - h_i, \cdots)$ in equation 6.28. In the actual implementation, this is done by evaluating the circuit dependency graph as a DAG as will be explained in the next section. This requires that the DAG evaluation be part of the damped Newton-Raphson Algorithm. The algorithm is simple and is outlined in Fig. 6.23.

Worth mentioning that the proposed method for simulator mode allows us to specify Newton-Raphson constraints in one module level and inherit it in a higher level module during DC simulation. This represent an explicit form of hierarchical knowledge reuse that is the main target of our proposed method.

6.6 Top-Down Evaluation of Dependency Graphs

The proposed methods shows how to construct module dependency graphs, in a bottom-up fashion, for both the designer and simulator modes. Since the knowledge stored in the module dependency graph is a directed acyclic graph (DAG), it can be divided into successive computational levels. The subsequent subsections will describe simple algorithms used to perform this division.

Graph-Based Newton-Raphson Algorithm

1 Verify that the number of equations = the number of input parameters
2 Evaluate the DAG for the input parameter vector $\mathbf{x}_k$
3 Get $\mathbf{F}_{KCL}(\mathbf{x}_k)$, $\mathbf{F}_{offset}(\mathbf{x}_k)$ and $\mathbf{F}_{feedback}(\mathbf{x}_k)$
4 Set $\mathbf{F}(\mathbf{x}_k) = [\mathbf{F}_{KCL}(\mathbf{x}_k) \quad \mathbf{F}_{offset}(\mathbf{x}_k) \quad \mathbf{F}_{feedback}(\mathbf{x}_k)]^T$
5 For each input parameter x_k^i in $\mathbf{x}_k$
6 Set the input parameter x_k^i equals to $x_k^i - h_i$
7 Evaluate the DAG for $x_k^i - h_i$
8 Get $\mathbf{F}_{KCL}(\cdots, x_k^i - h_i, \cdots)$, $\mathbf{F}_{offset}(\cdots, x_k^i - h_i, \cdots)$ and $\mathbf{F}_{feedback}(\cdots, x_k^i - h_i, \cdots)$
9 Compute the jacobian matrix $\mathbf{J}(\mathbf{x}_k)$ using equations 6.27 and 6.28
10 Restore the input parameter x_k^i changed at step (5)
11 End For
12 Solve $\mathbf{J}(\mathbf{x}_k) \cdot \mathbf{\Delta x}_k = -\mathbf{F}(\mathbf{x}_k)$ to get $\mathbf{\Delta x}_k$ using LU Factorization
13 Get next estimate $\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma \cdot \mathbf{\Delta x}_k$
14 If maximum iteration count is reached, then restore $\mathbf{x}_k$ and goto step (16)
15 Repeat 2-13 until $\mid \mathbf{x}_{k+1} - \mathbf{x}_k \mid \leq \epsilon_{relative} \cdot \max(\mid \mathbf{x}_{k+1} \mid, \mid \mathbf{x}_k \mid) + \epsilon_{absolute}$
16 Evaluate the DAG for the input parameter vector x_{k+1}
17 End

Figure 6.23: *Pseudo-code of the graph-based Newton-Raphson algorithm.*

6.6.1 Node Coloring

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to each vertex of a graph so that adjacent vertices are not assigned the same color. A computational level consists of the set of nodes that can be evaluated simultaneously. In our case, each computational level will be assigned one distinct color that is different from previous and successive computational levels. Coloring a computational level consists of coloring all nodes in this level with the same color. This means that nodes in the same computational level cannot be adjacent. The algorithm used for coloring is a variant of the *as-late-as-possible (ALAP) scheduling* algorithm [Kung85]. This is presented in the subsequent subsection.

6.6.2 Scheduling using As-Late-As-Possible Scheme (ALAP)

The ALAP scheduling assigns each node to the latest possible computational level. The algorithm is outlined in Fig. 6.24.

In this algorithm, the graph is checked if there exists nodes that do not possess outgoing arcs. These nodes are colored using the same color since they form one distinct computational level. Then all the arcs incident to these nodes are removed from the graph. This arc removal will create

```
ALAP Scheduling Algorithm
1     Given
2         V is the set of vertices,
3         Nodes u, v ∈ V,
4         A is the set of arcs,
5         Arc uv ∈ A,
6         Graph G = (V, A),
7         Color c
8
9     Set color c to 0
10    While not all v ∈ V in G = (V, A) are colored
11    do
12        For all uncolored v ∈ V in G
14            If no arcs are out of v then
15                Colorize v with color c
16                Remove all arcs uv ∈ A incident on v in G
17            End If
18        End For
19        Increment color c
20    End While
```

Figure 6.24: *Pseudo-code for the ALAP scheduling*.

another set of uncolored nodes which constitutes another computational level and so on. The algorithm stops when all graph nodes are colored.

6.6.3 Dependency Graph Evaluation

At the beginning, all the graph nodes will be scheduled into computational levels. Each computational level will then be visited in sequence. At each level, all nodes will be evaluated. Note that all graph nodes in one level can be evaluated simultaneously. The evaluation proceeds from a level to the next one, in a top-down (or left-to-right) approach till reaching the last computational level. At the end, all the parameter values have been propagated in the whole graph.

6.7 Putting all together

The SYNTHESIZE routine is outlined for modules in Fig. 6.25. When synthesizing a module, the steps depicted at lines 8-14 are executed:

1. In lines 2-4, the routine is called recursively for all children generators

```
1    function synthesize( generator )
2        for every child of generator
3            call synthesize(child)
4        end for
5        if generator is a device
6            generate dependencies for the reference transistor
7            eliminate all redundant dependencies
8        else if generator is a module
9            merge dependencies of all children generators
10           eliminate all redundant dependencies
11           if generator is the root generator then
12               resolve all external conflicts
13           end if
14       end if
15   end function
```

Figure 6.25: *Pseudo-code of the* SYNTHESIZE *routine*.

2. In line 5, the generator is checked if it represents a device level.

3. In line 6, the reference transistor dependencies are generated as depicted in subsection 6.4.2.

4. In line 7, the redundant dependencies are eliminated in devices as explained in appendix D.6.

5. In line 8, the generator is checked if it represents a module level.

6. In line 9, if it is a module then the dependency graphs of all its children generators are merged using the technique described in subsection 6.4.3.

7. In line 10, the redundant dependencies are eliminated from the resulting module dependency graph, as explained in appendix D.7.

8. In lines 11-13, if the current level is the root level generator, the over-specified designs are detected and resolved as described in subsection 6.5.2.

6.8 Detailed Example: Single-ended Two-Stage Amplifier

The sizing and biasing for the two-stage amplifier, shown in Fig. 6.26, will be studied for different designer's hypotheses. This will be performed in both *the designer mode* and *the simulator*

mode. Two main design problems will be investigated, namely: *the systematic offset* and *the negative feedback*.

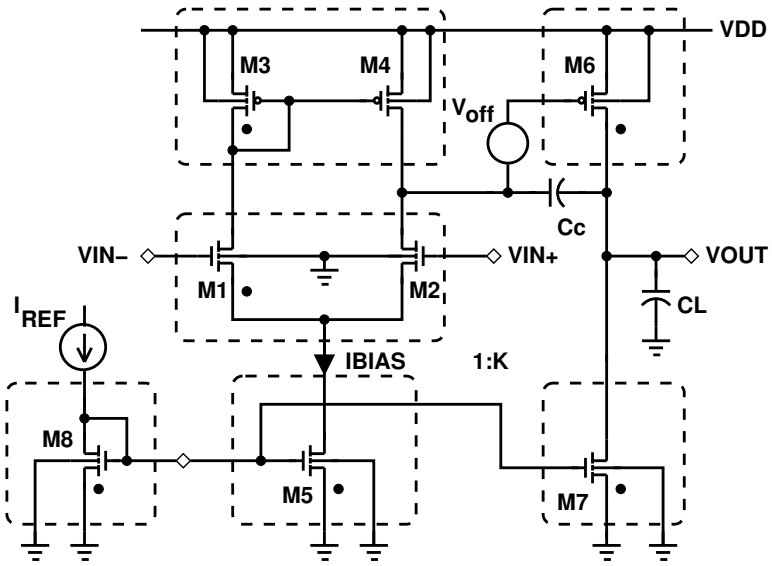

Figure 6.26: *Single-ended two-stage amplifier*.

6.8.1 Creating Amplifier Dependency Graphs in Designer Mode

The SYNTHESIZE routine outlined in Fig. 6.25 is applied in designer mode. The first step is to synthesize each device separately. Then, the different device dependency graphs will be merged to form the module dependency graph for the amplifier. These steps will be illustrated in further details in the next subsections.

Let us suppose that the module parameters of the amplifier are:

- $TEMP$: Temperature
- V_{DD}: Positive supply
- V_{SS}: Negative supply
- I_{BIAS}: Biasing current of the amplifier
- $V_{eg,M5}$: Overdrive voltage of biasing transistor M_5
- $V_{eg,CM}$: Overdrive voltage of the current mirror CM
- $V_{eg,DP}$: Overdrive voltage of the differential pair DP
- V_{INCM}: Common-mode input voltage

- $K = \frac{I_{M6}}{I_{BIAS}}$: Current ratio between the first and second stage
- L_{M_5,M_7,M_8}: Lengths of M_5, M_7 and M_8 which are equal
- L_{DP}: Length of the differential pair
- L_{CM} and L_{M_6}: Lengths of CM and M_6. They are equal in the case of minimum systematic offset ($L_{CM} = L_{M_6} = L_{CM,M_6}$)
- V_{OUTCM}: Common-mode output voltage

6.8.1.1 Synthesizing Children Devices

Applying steps 2-7 of the SYNTHESIZE routine, we generate the device dependency graph for each device in the amplifier.

The current mirror (M_3,M_4): The known parameters for the current mirror are: $Temp$, $V_{eg,CM}$, L_{CM}, $I_{DS,CM} = \frac{-I_{BIAS}}{2}$ and $V_{S,CM} = V_{B,CM} = V_{DD}$. Since the minimum systematic offset will be first studied, we set $L_{CM} = L_{CM,M_6}$. Synthesizing the current mirror using this set of parameters, we get the device dependency graph of Fig 6.27. Since the reference transistor M_3 is diode-connected, the device dependency graph consists mainly of the operator $OPVGD(V_{eg,CM})$.

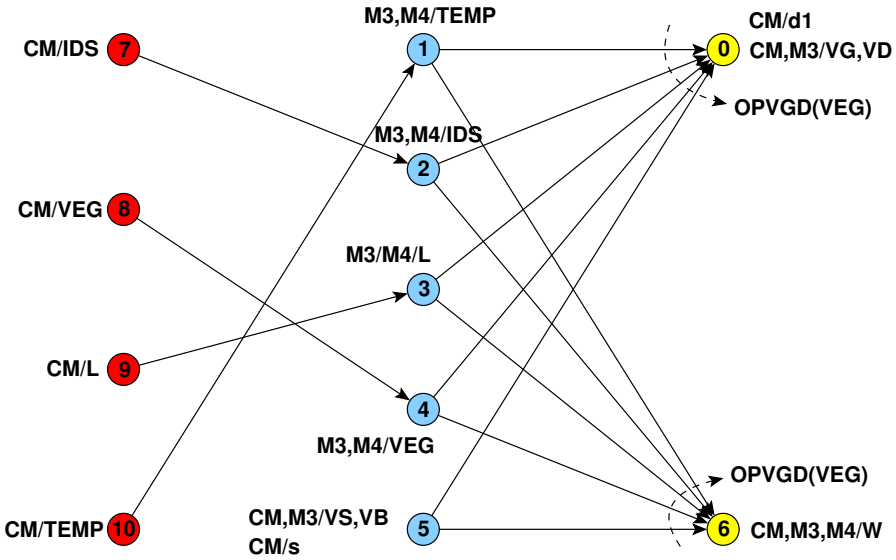

Figure 6.27: *Device dependency graph for the current mirror (M_3,M_4).*

The differential pair (M_1,M_2): The known parameters of the differential pair are: $Temp$, $V_{eg,DP}$, L_{DP}, $I_{DS,DP} = \frac{I_{BIAS}}{2}$, $V_G = V_{INCM}$ and $V_{B,DP} = V_{SS}$. The drain voltage $V_{D,DP}$ is the result of

computation of the current mirror. This is identified as the equipotential ($V_{D,DP}$,$V_{G/D,CM}$) using the method of equipotentials discussed in section 6.4.1. Synthesizing the differential pair, we get the device dependency graph of Fig. 6.28. Since the source voltage is unknown for the reference transistor M_1 and M_1 is not bulk-source connected, the device dependency graph consists mainly of the operator $OPVS(V_{eg,DP}, V_{B,DP})$.

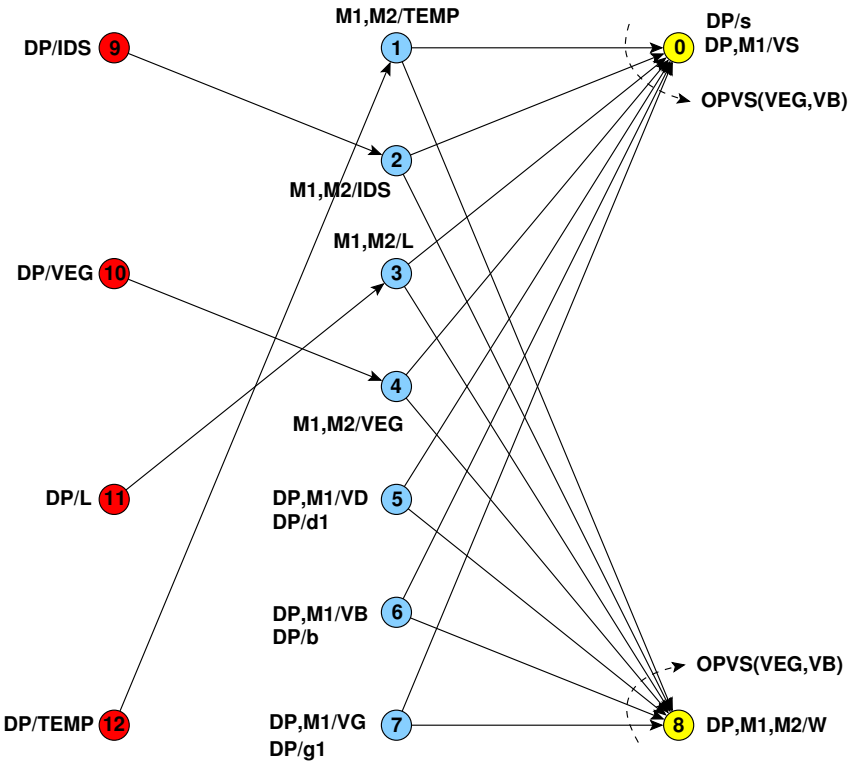

Figure 6.28: *Device dependency graph for the differential pair (M_1,M_2).*

The biasing transistor M_5: The known parameters of the biasing transistor M_5 are: $Temp$, V_{eg,M_5}, $L_{M_5} = L_{M_5,M_7,M_8}$, $I_{DS,M_5} = I_{BIAS}$ and $V_{S,M_5} = V_{B,M_5} = V_{SS}$. The drain voltage V_{D,M_5} is the result of computation of the differential pair. This is identified as the equipotential (V_{D,M_5},$V_{S,DP}$) using the method of equipotentials discussed in section 6.4.1. Synthesizing the biasing transistor, we get the device dependency graph of Fig. 6.29. Since the gate voltage of the reference transistor is unknown and M_5 is bulk-source connected, the device dependency graph consists mainly of the operator $OPVG(V_{eg,M_5})$.

The second stage load transistor M_6: The known parameters of the load transistor M_6 are: $Temp$, L_{M_6}, $I_{DS,M_6} = -K \cdot I_{BIAS}$, $V_{S,M_6} = V_{B,M_6}$. Since the minimum systematic offset will be first studied, then we impose the constraints that $L_{M_6} = L_{CM,M_6}$ and $V_{G,M_6} = V_{D,DP}$. The

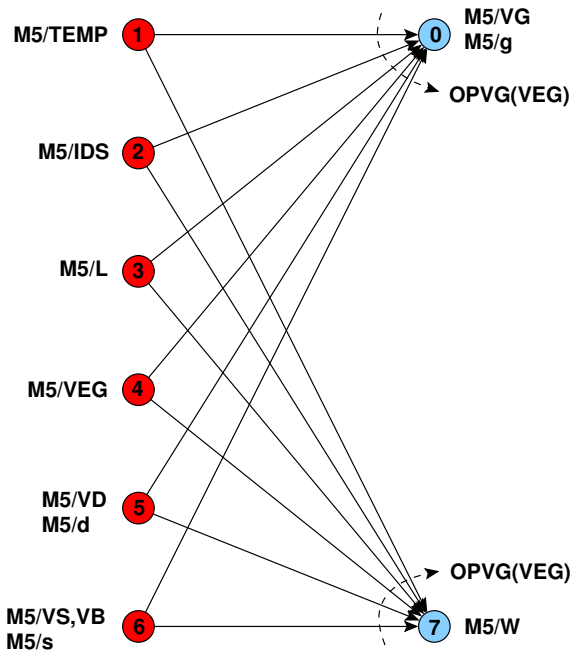

Figure 6.29: *Device dependency graph for the transistor* M_5.

constraint $V_{G,M_6} = V_{D,DP}$ makes V_{G,M_6} known since $V_{D,DP} = V_{G/D,CM}$ and $V_{G/D,CM}$ is computed from the current mirror . Synthesizing the load transistor M_6, we get the device dependency graph of Fig. 6.30. Since the width W_{M_6} is the only remaining unknown parameter, the device dependency graph consists mainly of the operator $OPW(V_{G,M_6}, V_{S,M_6})$.

The second stage biasing transistor M_7**:** The only known parameters are: $Temp$, $L_{M_7} = L_{M_5,M_7,M_8}$, $I_{DS,M_7} = K \cdot I_{BIAS}$, $V_{S,M_7} = V_{B,M_7}$. Since the gate voltage $V_{G,M_7} = V_{G,M_5}$ as the equipotential ($V_{G/D,M_8}$,V_{G,M_5}, V_{G,M_7}) is identified and V_{G,M_5} is previously computed from M_5, then V_{G,M_7} is also a known parameter. Synthesizing the biasing transistor M_7, we get the device dependency graph of Fig. 6.31. Since the width W_{M_7} is the only remaining unknown parameter, the device dependency graph consists mainly of the operator $OPW(V_{G,M_7}, V_{S,M_7})$.

The biasing circuit transistor M_8**:** The parameters of the biasing transistor M_8 are imposed identically to M_5. The known parameters are $Temp$, $L_{M_8} = L_{M_5,M_7,M_8}$, $V_{S,M_8} = V_{B,M_8}$. As the equipotential ($V_{G/D,M_8}$,V_{G,M_5}, V_{G,M_7}) is identified, then $V_{G/D,M_8}$, becomes a known parameter too. The unknown parameters are the width W_{M_8} and the current I_{DS,M_8}. Synthesizing the biasing transistor M_8, we get the device dependency graph of Fig. 6.32. Since we have two unknowns, the directed cycle (0,5) will be created. As we impose the constraint $W_{M_8} = W_{M_5}$, W_{M_8} becomes known resolving the directed cycle leaving I_{DS,M_8}as the only unknown to be computed. Since the

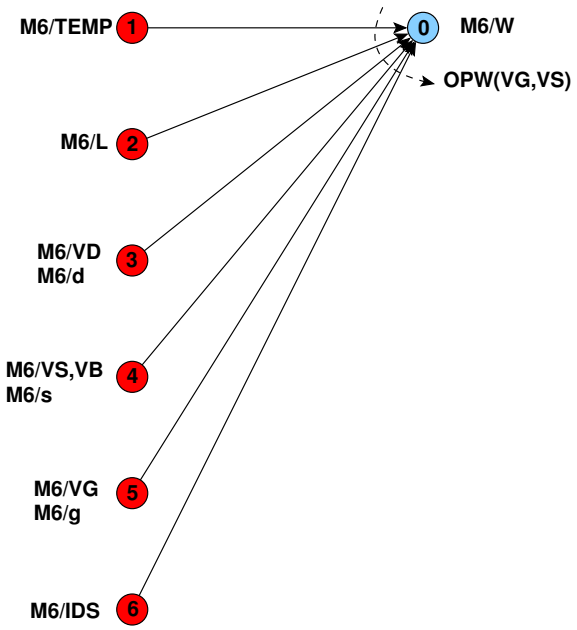

Figure 6.30: *Device dependency graph for the transistor M_6.*

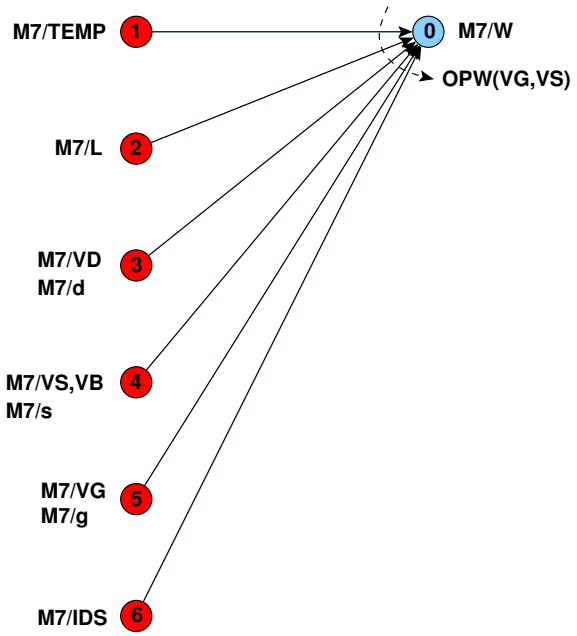

Figure 6.31: *Device dependency graph for the transistor M_7.*

current I_{DS,M_8} is the only remaining unknown parameter, the device dependency graph consists mainly of the operator $OPIDS(V_{G,M_8}, V_{S,M_8})$.

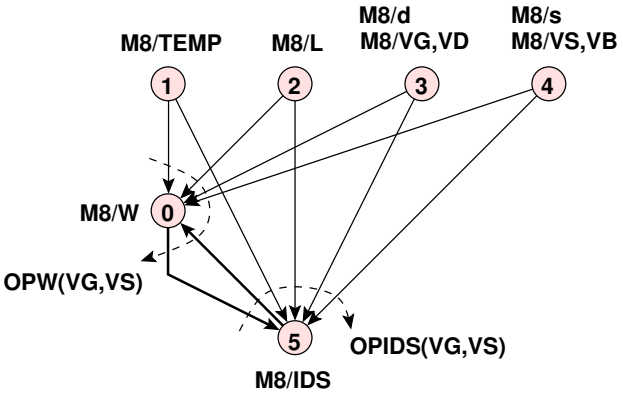

Figure 6.32: *Device dependency graph for the transistor M_8.*

6.8.1.2 Dependency Graph Without Systematic Offset in Designer Mode

To synthesize the amplifier, the module parameters $TEMP$, V_{DD}, V_{SS}, I_{BIAS}, $V_{eg,M5}$, $V_{eg,CM}$, $V_{eg,DP}$, K, L_{CM,M_6}, L_{M_5,M_7,M_8}, L_{DP}, V_{INCM} and V_{OUTCM} are set by the designer. To achieve minimum systematic offset:

- The constraint $V_{G,M_6} = V_{D,DP}$ (or $V_{G,M_6} = V_{D,M_1} = V_{D,M_2}$) is imposed for minimum systematic offset.

- The constraint $L_{M_6} = L_{CM} = L_{CM,M_6}$ is imposed to keep the same threshold voltage for the current mirror CM and M_6.

In addition, the constraint $W_{M_8} = W_{M_5}$ and $L_{M_8} = L_{M_5} = L_{M_7}$ are imposed. As the biasing current I_{BIAS} is set by the designer, the reference polarization current is computed for M_8. The SYNTHESIZE routine is executed for the module level. At that time, all the device dependency graphs are merged to form the module dependency graph illustrated in Fig. 6.33.

The module dependency graph for the amplifier possesses lots of characteristics:

1. The graph is a directed acyclic graph (DAG) as it evolves in only one left-to-right (or top-down) direction.

2. The rectangle nodes are the minimal set of design parameters that the designer should set for this graph. These parameters are (C1,I_{BIAS},42), (C1,$V_{eg,CM}$,62), (C1,L_{CM,M_6},65), (C1,$Temp$,67), (C2,V_{DD},58), (C2,$V_{eg,DP}$,63), (C2,L_{DP},66), (C3,V_{SS},57), (C3,V_{INCM},60), (C4,V_{eg,M_5},61), (C4,L_{M_8,M_5,M_7},64), (C5,K,43) and (C5,V_{OUTCM},59).

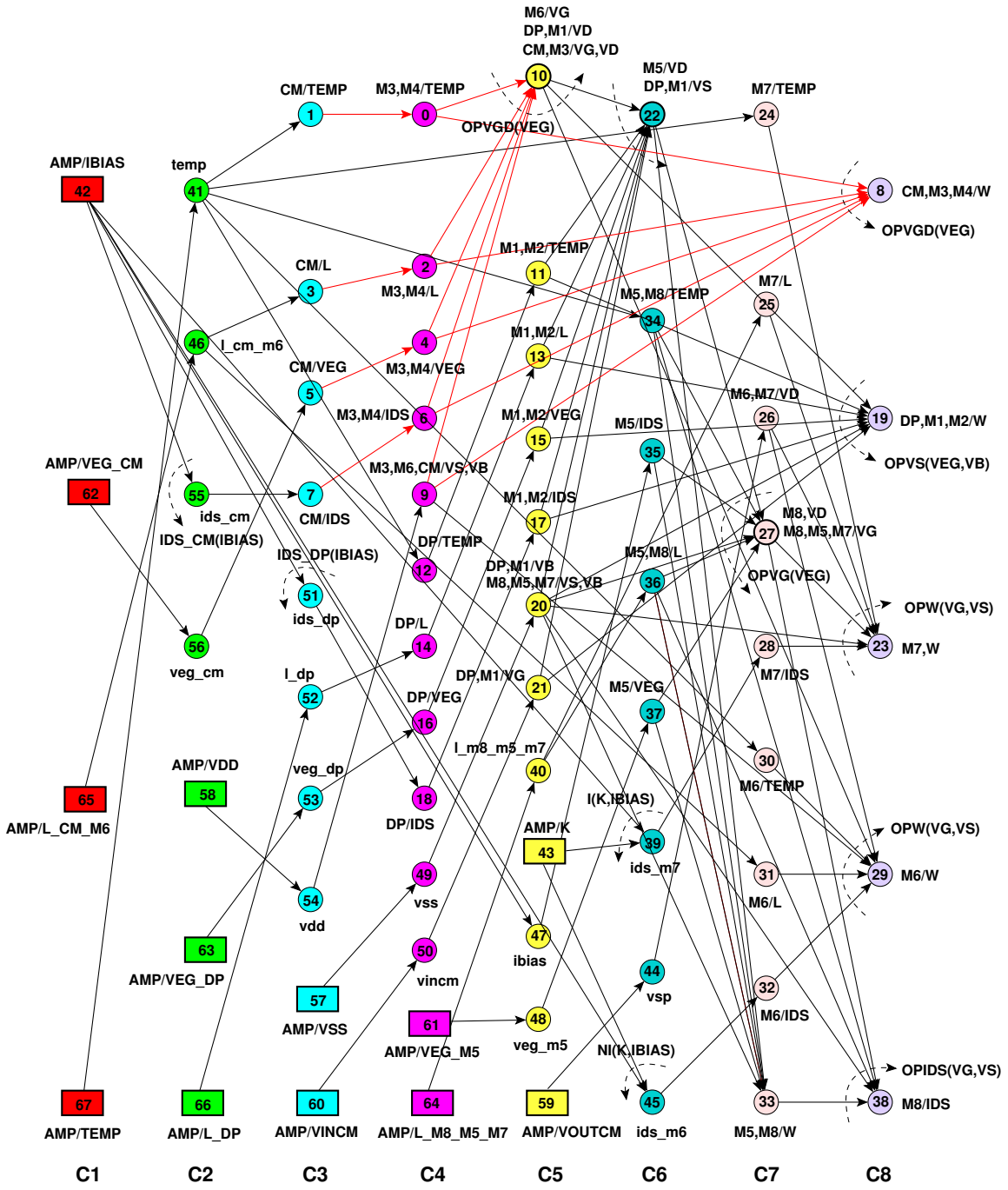

Figure 6.33: *Module dependency graph of the amplifier without systematic offset in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). The current mirror dependencies are represented by the red arcs. Device connectors, equipotentials and weights are not shown for clarity.*

3. The variables and parameters used for parameter mapping are represented as *fine circle* nodes. As an example, variable (C2,veg_cm,56) maps the amplifier parameter (C1,$V_{eg,CM}$,62) into the current mirror parameter (C3,$V_{eg,CM}$,5).

4. Device parameters are propagated to transistors forming the device. As an example, the current mirror parameter (C3,$V_{eg,CM}$,5) is propagated to M_3 via (C4,V_{eg,M_3},4) and to M_4 via (C4,V_{eg,M_4},4). Note that M_3 and M_4 share the same effective gate-source voltage (C4,$\{V_{eg,M_3}, V_{eg,M_4}\}$,4).

5. Since V_{eg,M_3} is specified, $V_{G/D,M_3}$ is computed by the operator *OPVGD(V_{eg,M_3})* in (C5,$V_{G/D,M_3}$,10). Operators are applied to *bold* nodes.

6. Transistor widths are computed in nodes (C8,$\{W_{M_3}, W_{M_4}\}$,8), (C8,$\{W_{M_1}, W_{M_2}\}$,19), (C8,W_{M_7},23), (C8,W_{M_6},29) and (C7,$\{W_{M_5}, W_{M_8}\}$,33).

7. The unknown reference current of I_{REF,M_8} is computed in node (C8,I_{DS,M_8},38) via the operator $OPIDS(V_{G,M_8}, V_{S,M_8})$.

8. The graph is divided into eight successive computational levels.

9. All nodes in one computational level are computed simultaneously.

10. The design plan presented in the graph appears as the following sequence of operators:

    (a) For the current mirror, $OPVGD(V_{eg,M3})$ computes (W_{M_3},$W_{M,4}$) in node (C8,$\{W_{M_3}, W_{M_4}\}$,8) and $V_{G/D,M_3}$ in node (C5,$V_{G/D,M_3}$,10).

    (b) Since $V_{D,M_1} = V_{G/D,M_3}$ for the differential pair, this is used by $OPVS(V_{eg,M_1}, V_{B,M_1})$ to compute (W_{M_1}, W_{M_2}) in node (C8,$\{W_{M_1}, W_{M_2}\}$,19) and V_{S,M_1} in node (C6,V_{S,M_1},22).

    (c) Since $V_{S,M_1} = V_{D,M_5}$ for transistor M_5, $OPVG(V_{eg,M_5})$ computes $V_{G,M_5} = V_{G/D,M_8} = V_{G,M_7}$ and W_5 in nodes (C7,$\{V_{G,M_5}, V_{G,M_7}, V_{G/D,M_8}\}$,27) and (C7,$\{W_{M_5}, W_{M_8}\}$,33) respectively.

    (d) Since $V_{G/D,M_8} = V_{G,M_5}$ for transistor M_8, $OPIDS(V_{G,M_8}, V_{S,M_8})$ computes I_{DS,M_8} in node (C8,I_{DS,M_8},38).

    (e) For transistor M_6, $OPW(V_{G,M_6}, V_{S,M_6})$ computes W_{M_6} in node (C8,W_{M_6},29).

    (f) For transistor M_7, $OPW(V_{G,M_7}, V_{S,M_7})$ computes W_{M_7} in node (C8,W_{M_7},23).

11. The minimum systematic offset hypothesis $V_{D,M_1} = V_{D,M_2} = V_{G,M_6}$ appears as node (C5,$V_{G/D,M_3}$,10) where V_{D,M_1} and V_{G,M_6} share the same node.

12. The constraint $L_{CM} = L_{M_6}$ appears as one parameter in node (C1,L_CM_M6,65) that affects node (C2,l_cm_m6,46). This, in turn, affects both nodes (C3,L_{CM},3) and (C7,L_{M6},31).

13. The constraint $L_{M_8} = L_{M_5} = L_{M_7}$ appears as one parameter in node (C4,L_{M_8,M_5,M_7},64) that affects node (C5,$l\_m8\_m5\_m7$,40). This, in turn, affects both nodes (C6,L_{M_8,M_5},36) and (C7,L_{M_7},25).

14. The constraint $W_{M_8} = W_{M_5}$ appears as node (C7,$\{W_{M_8}, W_{M_5}\}$,33).

15. A designer-defined procedure is used at node (C2,ids_cm,55) to compute $I_{DS,CM}$ from (C1,I_{BIAS},42).

The amplifier dependency graph is evaluated by setting its parameters as shown in Table 6.2.

Table 6.2: *Input Parameters for Minimum Systematic Offset in Designer Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	I_{BIAS}(μA)	30.0
$V_{eg,M5}$(V)	0.1	$L_{M5,M7,M8}$(μm)	0.34
$V_{eg,CM}$(V)	-0.12	L_{DP}(μm)	0.34
$V_{eg,DP}$(V)	0.12	$L_{CM,M6}$(μm)	0.34
V_{INCM}(V)	0.6	V_{OUTCM}(V)	0.6
$K = \frac{I_{M6}}{I_{BIAS}}$	5.0		

The amplifier dependency graph is then evaluated and the DC operating is computed in $0.13\mu m$ technology. Then the amplifier is simulated as a unity buffer closed-loop configuration. The synthesis results are compared against the simulation results in Table 6.3. The results show that the proposed methodology sized and biased the amplifier with very acceptable precision. The slight differences between the simulation and synthesis comes essentially from the fact that the synthesis phase is coupled with a layout generation phase into which the widths are aligned with the physical grid.

Table 6.4 shows the computed widths and the number of fingers for all the devices of the amplifier. In addition, the reference current I_{DS,M_8} is also computed from the graph.

6.8.1.3 Dependency Graph With Systematic Offset in Designer Mode

To synthesize the amplifier, the module parameters $TEMP$, V_{DD}, V_{SS}, I_{BIAS}, $V_{eg,M5}$, V_{eg,M_6}, $V_{eg,CM}$, $V_{eg,DP}$, K, L_{CM}, L_{M_6}, L_{M_5,M_7,M_8}, L_{DP}, V_{INCM} and V_{OUTCM} are set by the designer. To generate a conflict between the two stages of the amplifier:

1. L_{CM} and L_{M_6} are specified separately to account for different threshold voltages in the current mirror CM and transistor M_6.

2. The constraint $V_{G,M_6} = V_{G/D,CM}$ is imposed.

Table 6.3: *Operating Point Results for Minimum Systematic Offset in Designer Mode.*

Parameter	Synthesis		Simulation		
	M_1,M_2	M_6	M_1	M_2	M_6
$I_{DS}(\mu A)$	15.0	-150.0	15.004	15.003	-149.83
$V_{GS}(V)$	0.481564	-0.473856	0.48142	0.48142	-0.47376
$V_{DS}(V)$	0.607707	-0.6	0.60753	0.60765	-0.6
$V_{BS}(V)$	-0.118436	0.0	-0.11858	-0.11858	0.0
$V_{th}(V)$	0.361564	-0.353809	0.36167	0.36167	-0.35381
$V_{eg}(V)$	0.12	Not Given	0.11975	0.11975	-0.11995
$V_{dsat}(V)$	0.119121	-0.119337	0.11897	0.11897	-0.11928
$g_m(mA/V)$	0.190776	1.90276	0.19106	0.19106	1.9015
$g_{ds}(\mu A/V)$	2.27195	17.756	2.2738	2.2736	17.738
$g_{mb}(mA/V)$	0.0356447	0.369543	0.035701	0.0357	0.36929
$C_{gd}(fF)$	1.03858	24.5575	1.0422	1.0422	24.558
$C_{gs}(fF)$	5.30485	152.293	5.3216	5.3215	152.26

Table 6.4: *Computed Parameters for Minimum Systematic Offset in Designer Mode.*

$(W/M)_{DP}$	$(W/M)_{CM}$	$(W/M)_{M_5}$	$(W/M)_{M_6}$	$(W/M)_{M_7}$	$(W/M)_{M_8}$	I_{DS,M_8}
$2.08\mu m/4$	$6.32\mu m/2$	$6.64\mu m/4$	$62.2\mu m/20$	$27.6\mu m/16$	$6.64\mu m/4$	$35.29\mu A$

3. In addition, $V_{eg,CM}$ and V_{eg,M_6} are specified differently to generate the conflict as explained in section 6.5.2.

After synthesizing the amplifier using the above hypotheses, we get the amplifier dependency graph shown in Fig. 6.34. Comparing the graph with the case of minimum systematic offset, we notice the introduction of the new offset node (C8,V_{OFFSET},0). Node (C5,$\{V_{G,M_6}, V_{D,DP}, V_{D,M_1}, V_{G/D,CM}, V_{G/D,M_3}\}$,10) of Fig. 6.33 has been split to the two nodes (C5,$\{V_{D,DP}, V_{D,M_1}, V_{G/D,CM}, V_{G/D,M_3}\}$,2) and (C7,$V_{G,M_6}$,1) of Fig. 6.34. This is done to resolve the conflict caused between these two nodes.

The amplifier dependency graph is then evaluated using the parameter values listed in Table 6.5. Note that $L_{CM} \neq L_{M6}$ and that $V_{eg,M6} \neq V_{eg,CM}$ to account for systematic offset hypotheses listed above.

The amplifier is then simulated in a unity buffer closed-loop configuration. Table 6.6 shows the computed and simulated operating points, as well as, the computed input offset. The synthesis part of the table allows us to compute the internal offset of the amplifier and bring it to the positive input of the amplifier. Using equation (6.2), the input offset is found to be -0.46805 mV. More accurate equations from OCEANE [Porte08] have been used for the static gain of the first stage and it was found that the input offset equals -0.471139 mV. This offset balances the amplifier

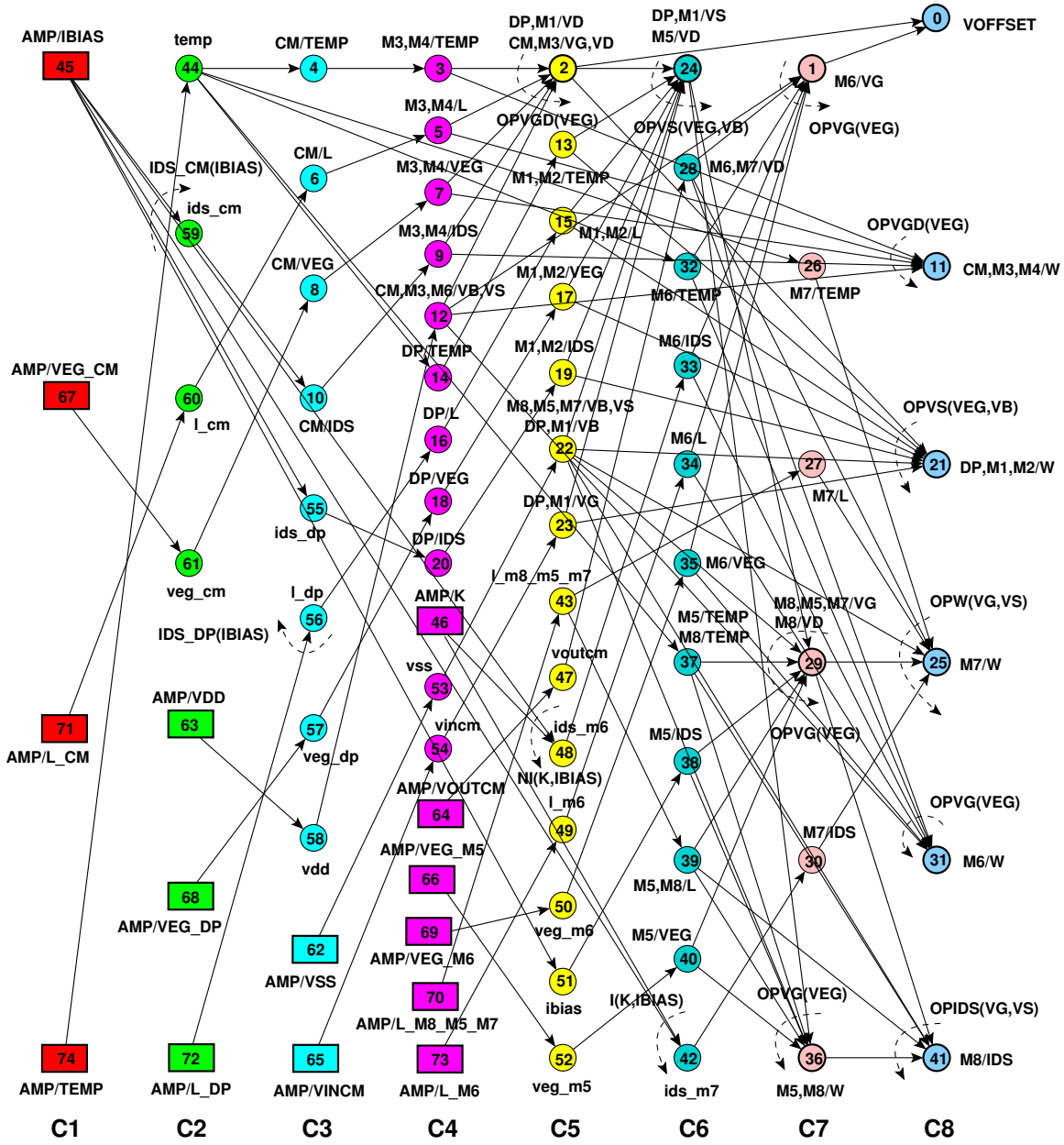

Figure 6.34: *Module dependency graph of the amplifier with systematic offset in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

during simulation. Consequently, the common-mode output voltage $V_{OUTCM} = V_{DD} + V_{DS,M_6}$ is at 0.6V. Notice that the simulated operating points of M_1 and M_2 are slightly different under the influence of the offset voltage that appears now at the input. Table 6.7 shows the computed parameters for the case of systematic offset. It is evident from the table that all dimensions and

Table 6.5: *Input Parameters For Systematic Offset in Designer Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{BIAS}(\mu$A)	30.0
$V_{eg,M5}$(V)	0.1	$L_{M5,M7,M8}(\mu$m)	0.34
$V_{eg,CM}$(V)	-0.12	$L_{DP}(\mu$m)	0.34
$V_{eg,DP}$(V)	0.12	$L_{CM}(\mu$m)	0.34
$V_{eg,M6}$(V)	-0.1	$L_{M6}(\mu$m)	0.34
V_{INCM}(V)	0.6	V_{OUTCM}(V)	0.6
$K = \frac{I_{M6}}{I_{BIAS}}$	5.0		

number of fingers are the same for the minimum systematic offset, except for W_{M_6} which has increased to compensate for the lower effective gate voltage.

Table 6.6: *Operating Point With Systematic Offset in Designer Mode.*

Parameter	Synthesis		Simulation		
	M_1,M_2	M_3,M_4	M_1	M_2	M_4
$I_{DS}(\mu A)$	15.0	-15.0	15.023	14.978	-14.978
$V_{GS}(V)$	0.481564	-0.473856	0.48150	0.48103	-0.47399
$V_{DS}(V)$	0.607707	-0.473856	0.60752	0.62737	-0.45413
$V_{th}(V)$	0.361564	-0.353856	0.36165	0.36165	-0.35386
$V_{eg}(V)$	0.12	-0.12	0.11985	0.11938	-0.12013
$g_m(mA/V)$	0.190776	0.190494	0.19121	0.191	0.1901
$g_{ds}(\mu A/V)$	2.27195	2.22754	2.2764	2.2362	2.3301
	M_6		M_6		
$I_{DS}(\mu A)$	-150.0		-150.16		
$V_{GS}(V)$	-0.454011		-0.45413		
$V_{DS}(V)$	-0.6		-0.6		
$V_{th}(V)$	-0.354011		-0.35401		
$V_{eg}(V)$	-0.1		-0.10012		
$g_m(mA/V)$	2.09537		2.0963		
$V_{i,off}(mV)$	-0.46805[1]		Not used for simulation		
$V_{i,off}(mV)$	-0.471139[2]		-0.471139		

1. Using equation 6.2.
2. Using more accurate equations from OCEANE [Porte08].

Table 6.7: *Computed Parameters for Systematic Offset*.

$(W/M)_{DP}$	$(W/M)_{CM}$	$(W/M)_{M_5}$	$(W/M)_{M_6}$	$(W/M)_{M_7}$	$(W/M)_{M_8}$	I_{DS,M_8}
2.08$\mu m/4$	6.32$\mu m/2$	6.64$\mu m/4$	80.63$\mu m/22$	27.6$\mu m/16$	6.64$\mu m/4$	35.29μA

Appendix C illustrates the CAIRO+ generator for the two-stage amplifier in the designer mode. It shows how to write the SIZE procedure of the amplifier module in the case of presence of systematic offset voltage.

6.8.2 Creating Amplifier Dependency Graphs in Simulator Mode

The simulator mode is intended to verify the synthesis results obtained in designer mode. In this section, we will go further beyond this objective. We will show how to use the simulator mode to deal with over-specified design problems, namely *the systematic offset*. It will be shown how to compute the systematic input offset of an analog circuit in different ways. In addition, it will be shown how to deal with negative feedback circuits in simulator mode.

6.8.2.1 Dependency Graph With Systematic Input Offset in Simulator Mode

The amplifier is assumed to be simulated in an open loop configuration as shown in Fig. 6.35. The output node and the negative amplifier input are fixed in potential in order to compute the systematic input offset at the positive input of the amplifier.

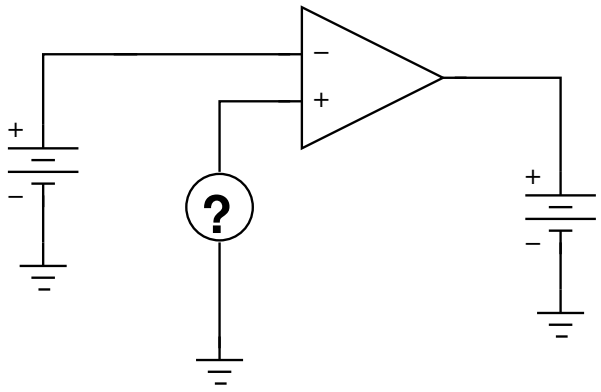

Figure 6.35: *Amplifier in open-loop configuration*.

To simulate the amplifier in simulator mode, the module parameters $Temp$, V_{DD}, V_{SS}, L_{M_1}, L_{M_2}, L_{M_3}, L_{M_4}, L_{M_5}, L_{M_6}, L_{M_7}, L_{M_8}, W_{M_1}, W_{M_2}, W_{M_3}, W_{M_4}, W_{M_5}, W_{M_6}, W_{M_7}, W_{M_8}, $V_{G/D,M_3}$, V_{OUT}, V_{INN} and I_{REF,M_8} are chosen. The number of fingers are set in the generator code. Note that in simulator mode, parameters mainly consists of widths, lengths and number of fingers exactly as for a standard simulator.

In addition, the Newton-Raphson constraint

$$F_{KCL}(V_{G/D,M_3}) = I_{DS,M_5}(V_{G/D,M_3}) - I_{DS,M_1}(V_{G/D,M_3}) - I_{DS,M_2}(V_{G/D,M_3}) = 0.0 \tag{6.29}$$

is added to ensure that the Kirchhoff's Current Law (KCL) is satisfied at the drain node of M_5. This is satisfied by solving for $V_{G/D,M_3}$ as one unknown. $V_{G/D,M_3}$ is selected as a parameter since it controls both currents I_{DS,M_1} and I_{DS,M_2}.

The resulting amplifier dependency graph is generated as shown in Fig. 6.36. The design plan represented by this graph is depicted as follows:

1. $V_{G/D,M_8}$ is computed in node (C4,$\{V_{G/D,M_8}, V_{G,M_5}, V_{G,M_7}\}$,27) using the operator $OPVGD(V_{S,M_8}, W_{M_8})$. Note that $V_{G/D,M_8} = V_{G,M_5} = V_{G,M_7}$.

2. Using V_{G,M_7}, I_{DS,M_7} is computed in node (C5,I_{DS,M_7},23) using the operator $OPIDS(V_{G,M_7}, V_{S,M_7})$. This is propagated to node (C6,I_{DS,M_6},30) which is used to compute $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$ in node (C7, $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$,3) using the operator $OPVG(V_{S,M_6}, W_{M_6})$.

3. Using the computed V_{D,M_2}, the gate node of M_2 is computed in node (C10,V_{G,M_2},11) using the operator $OPVG(V_{S,M_2}, V_{B,M_2}, W_{M_2})$. Note that this is the positive input of the amplifier to be computed.

4. The Newton-Raphson constraint node (C10,$F_{KCL}(V_{G/D,M_3})$,41) depends on the currents I_{DS,M_5}, I_{DS,M_1} and I_{DS,M_2} which are computed as follows:

   (a) I_{DS,M_5} is computed at node (C9,I_{DS,M_5},33) using the operator $OPIDS(V_{G,M_5}, V_{S,M_5})$.

   (b) I_{DS,M_3} is computed in node (C6,I_{DS,M_3},7) using the operator $OPIDS(V_{G,M_3}, V_{S,M_3})$. It is then propagated to node (C7,I_{DS,M_1},19) with a weight equal to 1.0.

   (c) I_{DS,M_4} is computed in node (C8,I_{DS,M_4},0) using the operator $OPIDS(V_{G,M_4}, V_{S,M_4})$. It is then propagated to node (C9,I_{DS,M_2},13) with a weight equal to 1.0.

   (d) The constraint solves for a new value for (C3,$V_{G/D,M_3}$,42) which is the unknown.

The graph is evaluated using the parameter values defines in Table 6.8. These parameter values are extracted from subsection 6.8.1.3 to verify that the systematic input offset computed for the designer mode was correct.

After evaluating the graph and solving for $V_{G/D,M_3}$, we get the results in Table 6.9. In this table, the systematic input offset is computed for the positive input terminal of the amplifier. The simulator mode results agrees to a good precision with the designer mode results in Table 6.6. This ensures how useful the simulator mode in verifying designer hypotheses and constraints.

6.8.2.2 Dependency Graph With Systematic Input Offset and Negative Feedback in Simulator Mode

The amplifier is assumed to be simulated in a closed loop configuration as shown in Fig. 6.37. The output node is fixed in potential. A negative feedback of the output to the negative amplifier input is connected. It is required to compute the common-mode input voltages at the positive and negative inputs of the amplifier. The systematic input offset will be the difference between the voltages at the positive and negative terminals.

To simulate the amplifier in simulator mode, the module parameters $Temp$, V_{DD}, V_{SS}, L_{M_1}, L_{M_2}, L_{M_3}, L_{M_4}, L_{M_5}, L_{M_6}, L_{M_7}, L_{M_8}, W_{M_1}, W_{M_2}, W_{M_3}, W_{M_4}, W_{M_5}, W_{M_6}, W_{M_7}, W_{M_8}, $V_{G/D,M_3}$, V_{OUT}, V_{INP}, V_{INN} and I_{REF,M_8} are chosen. The number of fingers are set in the generator code.

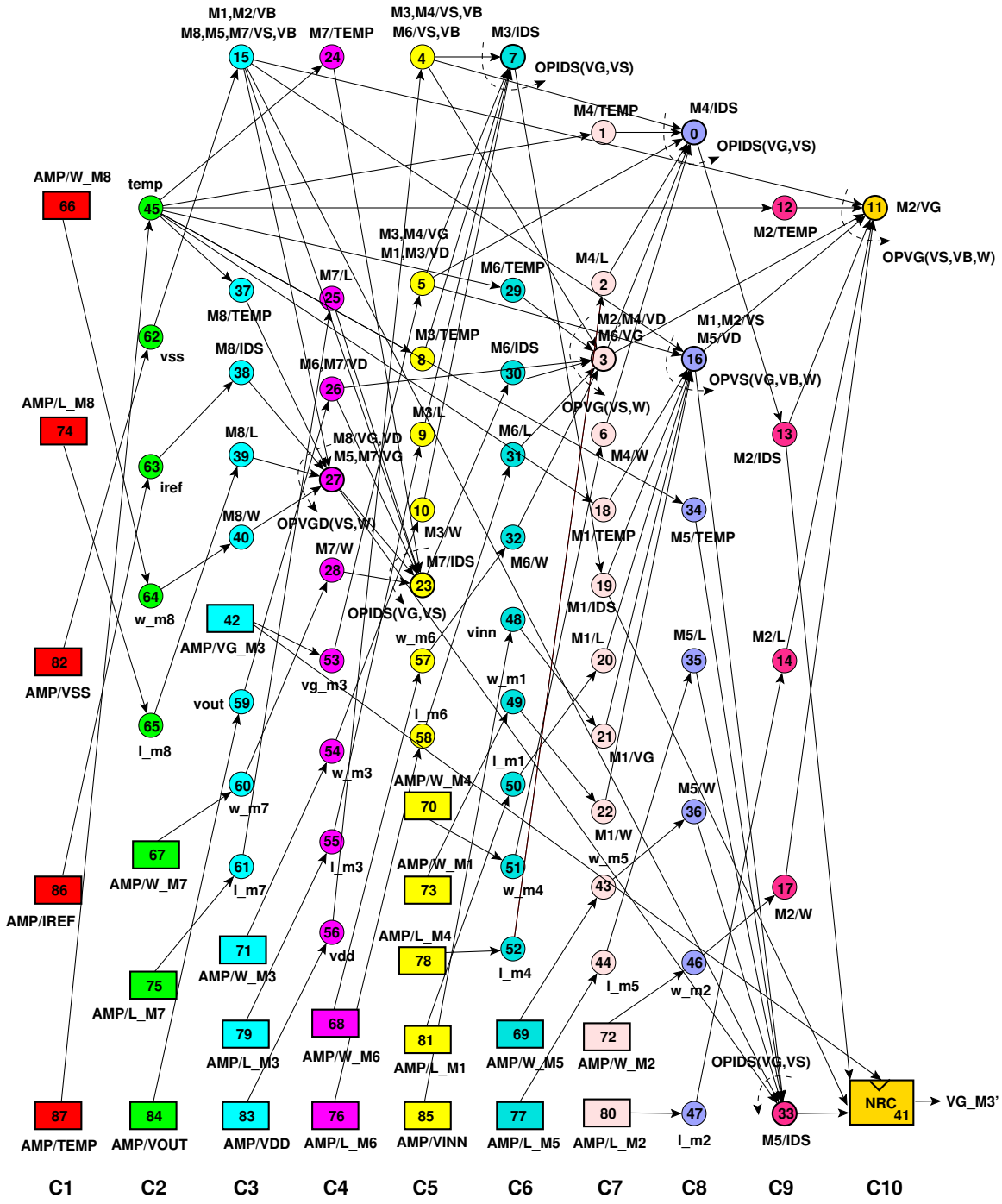

Figure 6.36: *Module dependency graph of the amplifier with systematic offset in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

Table 6.8: *Input Parameters For Systematic Offset in Simulator Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{REF,M_8}(\mu A)$	35.29
$(W/M)_{M_1}(\mu m)$	2.08/4	$L_{M_1}(\mu m)$	0.34
$(W/M)_{M_2}(\mu m)$	2.08/4	$L_{M_2}(\mu m)$	0.34
$(W/M)_{M_3}(\mu m)$	6.32/2	$L_{M_3}(\mu m)$	0.34
$(W/M)_{M_4}(\mu m)$	6.32/2	$L_{M_4}(\mu m)$	0.34
$(W/M)_{M_5}(\mu m)$	6.64/4	$L_{M_5}(\mu m)$	0.34
$(W/M)_{M_6}(\mu m)$	80.63/22	$L_{M_6}(\mu m)$	0.34
$(W/M)_{M_7}(\mu m)$	27.6/16	$L_{M_7}(\mu m)$	0.34
$(W/M)_{M_8}(\mu m)$	6.64/4	$L_{M_8}(\mu m)$	0.34
$V_{G/D,M_3}$(V)	unknown	V_{OUT}(V)	0.6
V_{INN}(V)	0.6		

Table 6.9: *Computed Parameters for Systematic Offset in Simulator Mode.*

$V_{G/D,M_3}$	$V_{INP} = V_{G,M_2}$	$V_{in,off} = V_{INP} - V_{INN}$
0.726004 V	0.599522 V	-0.478 mV

In addition, the Newton-Raphson constraint

$$F_{KCL}(V_{G/D,M_3}) = I_{DS,M_5}(V_{G/D,M_3}) - I_{DS,M_1}(V_{G/D,M_3}) - I_{DS,M_2}(V_{G/D,M_3}) = 0.0 \tag{6.30}$$

is added to ensure that the Kirchhoff's Current Law (KCL) is satisfied at the drain node of M_5. This is satisfied by solving for $V_{G/D,M_3}$ as one unknown.

To account for negative feedback, a Newton-Raphson constraint is added:

$$F_{feedback}(V_{INN}) = V_{G,M_1}(V_{INN}) - V_{D,M_7}(V_{INN}) = 0.0 \tag{6.31}$$

where the unknown is the negative input terminal voltage V_{INN}.

To bring the systematic offset to the positive input, a Newton-Raphson constraint is added:

$$F_{offset}(V_{INP}) = V_{S,M_1}(V_{INP}) - V_{S,M_2}(V_{INP}) = 0.0 \tag{6.32}$$

where the unknown is the positive input terminal voltage V_{INP}.

The resulting amplifier dependency graph is generated as shown in Fig. 6.38. Note that a conflict exists between the sources V_{S,M_1} and V_{S,M_2}. This gives rise to the introduction of an offset node (C12,V_{OFFSET},4). The design plan represented by this graph is depicted as follows:

1. $V_{G/D,M_8}$ is computed in node (C4,$\{V_{G/D,M_8}, V_{G,M_5}, V_{G,M_7}\}$,31) using the operator $OPVGD(V_{S,M_8}, W_{M_8})$. Note that $V_{G/D,M_8} = V_{G,M_5}, = V_{G,M_7}$

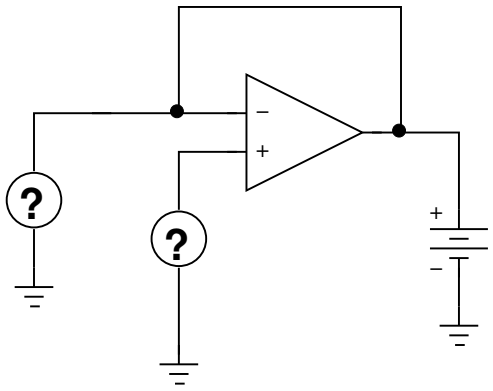

Figure 6.37: *Amplifier in closed-loop configuration with output fixed in potential.*

2. Using V_{G,M_7}, I_{DS,M_7} is computed in node (C5,I_{DS,M_7},27) using the operator $OPIDS(V_{G,M_7}, V_{S,M_7})$. This is equally propagated to node (C6,I_{DS,M_6},34) which is used to compute $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$ in node (C7, $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$,8) using the operator $OPVG(V_{S,M_6}, W_{M_6})$.

3. The Newton-Raphson constraint node (C12,$F_{offset}(V_{INP})$,0) depends on the voltages V_{S,M_1} and V_{S,M_2} which are computed as follows:

   (a) V_{S,M_1} is computed at node (C11,V_{S,M_1},2) using operator $OPVS(V_{G,M_1}, V_{B,M_1}, W_{M_1})$.

   (b) V_{S,M_2} is computed at node (C10,$\{V_{S,M_2}, V_{D,M_5}\}$,3) using operator $OPVS(V_{G,M_2}, V_{B,M_2}, W_{M_2})$.

   (c) The constraint solves for (C7,V_{INP},1) which is the unknown.

4. The Newton-Raphson constraint node (C12,$F_{KCL}(V_{G/D,M_3})$,45) depends on the currents I_{DS,M_5}, I_{DS,M_1} and I_{DS,M_2} which are computed as follows:

   (a) I_{DS,M_5} is computed at node (C11,I_{DS,M_5},37) using the operator $OPIDS(V_{G,M_5}, V_{S,M_5})$.

   (b) I_{DS,M_3} is computed in node (C9,I_{DS,M_3},12) using the operator $OPIDS(V_{G,M_3}, V_{S,M_3})$. It is then propagated to node (C10,I_{DS,M_1},23) with a weight equal to 1.0.

   (c) I_{DS,M_4} is computed in node (C8,I_{DS,M_4},5) using the operator $OPIDS(V_{G,M_4}, V_{S,M_4})$. It is then propagated to node (C9,I_{DS,M_2},17) with a weight equal to 1.0.

   (d) The constraint solves for (C5,$V_{G/D,M_3}$,46) which is the unknown.

5. The Newton-Raphson constraint node (C12,$F_{feedback}(V_{INN})$,47) depends on the voltages V_{G,M_1} and V_{D,M_7} which are computed as follows:

   (a) V_{G,M_1} is set via the unknown parameter (C8,V_{INN},48).

   (b) V_{D,M_7} is constant and is set via the parameter (C2,V_{OUT},75)

   (c) The constraint solves for (C8,V_{INN},48) which is the unknown.

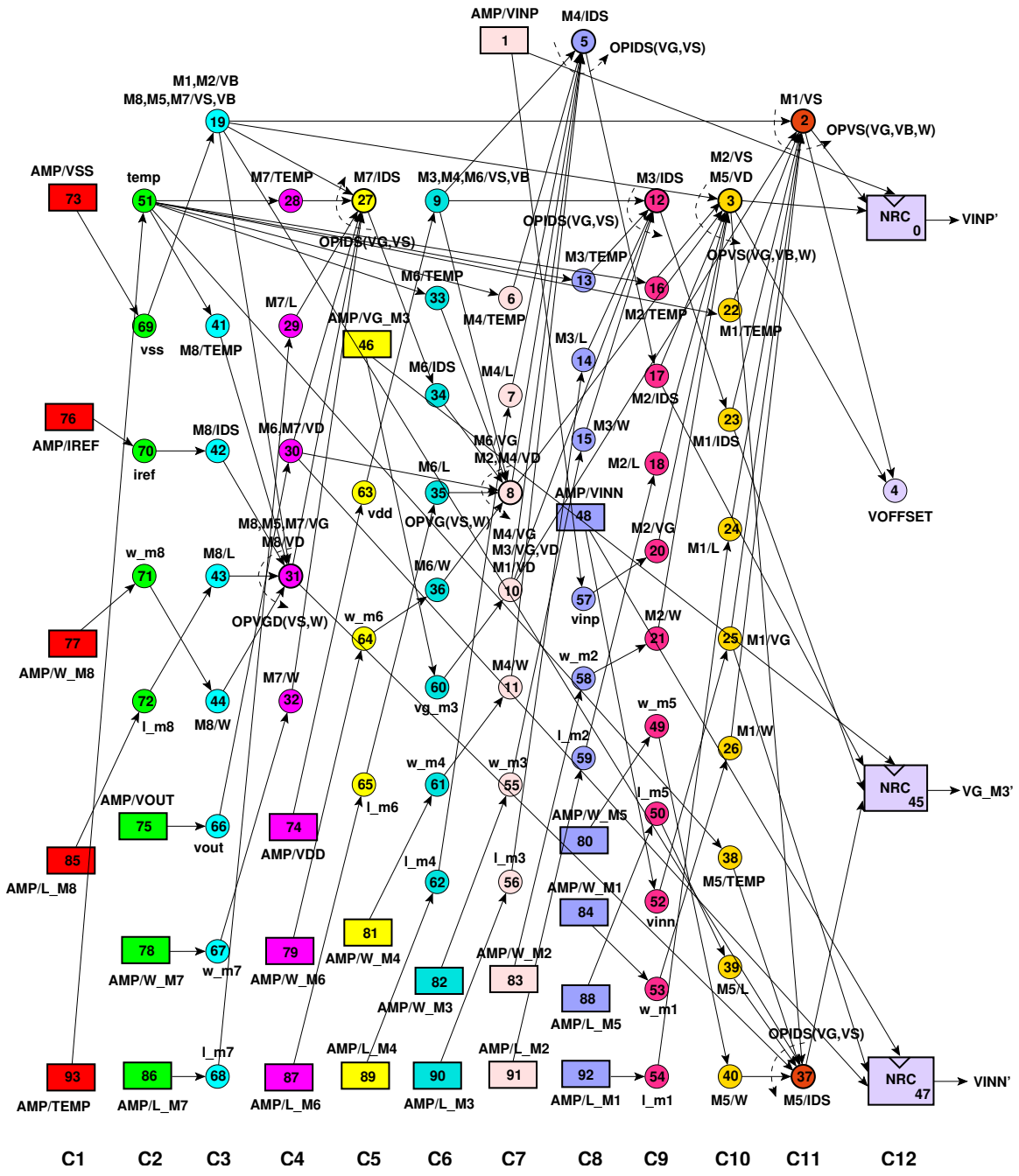

Figure 6.38: *Module dependency graph of the amplifier with systematic input offset and negative feedback in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

The graph is evaluated using the parameter values defines in Table 6.10. These parameter values are extracted from subsection 6.8.1.3.

Table 6.10: *Input Parameters For Systematic Input Offset and Negative Feedback in Simulator Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{REF,M_8}(\mu A)$	35.29
$(W/M)_{M_1}(\mu m)$	2.08/4	$L_{M_1}(\mu m)$	0.34
$(W/M)_{M_2}(\mu m)$	2.08/4	$L_{M_2}(\mu m)$	0.34
$(W/M)_{M_3}(\mu m)$	6.32/2	$L_{M_3}(\mu m)$	0.34
$(W/M)_{M_4}(\mu m)$	6.32/2	$L_{M_4}(\mu m)$	0.34
$(W/M)_{M_5}(\mu m)$	6.64/4	$L_{M_5}(\mu m)$	0.34
$(W/M)_{M_6}(\mu m)$	80.63/22	$L_{M_6}(\mu m)$	0.34
$(W/M)_{M_7}(\mu m)$	27.6/16	$L_{M_7}(\mu m)$	0.34
$(W/M)_{M_8}(\mu m)$	6.64/4	$L_{M_8}(\mu m)$	0.34
$V_{G/D,M_3}$(V)	unknown	V_{OUT}(V)	0.6
V_{INN}(V)	unknown	V_{INP}(V)	unknown

After evaluating the graph and solving for $V_{G/D,M_3}$, V_{INN} and V_{INP}, we get the results in Table 6.11. In this table, the computed systematic input offset is computed for the positive input terminal of the amplifier. The simulator mode results agree to a good precision with the designer mode results in Table 6.6.

Table 6.11: *Computed Parameters for Systematic Input Offset and Negative Feedback in Simulator Mode.*

$V_{G/D,M_3}$	V_{INP}	$V_{in,off} = V_{INP} - V_{INN}$
0.726004 V	0.599522 V	-0.478 mV

Appendix F illustrates the CAIRO+ generator for the two-stage amplifier in the simulator mode for the systematic input offset and negative feedback. It shows how to write the SIZE procedure in this case.

6.8.2.3 Dependency Graph With Systematic Output Offset and Negative Feedback in Simulator Mode

The amplifier is assumed to be simulated in a closed loop configuration as shown in Fig. 6.39. The output node is free in potential. A negative feedback of the output to the negative amplifier input is connected. It is required to compute the common-mode input voltages at the positive and negative inputs of the amplifier. The systematic input offset will be the difference between the output voltage (negative input terminal voltage) and the positive input terminal voltage.

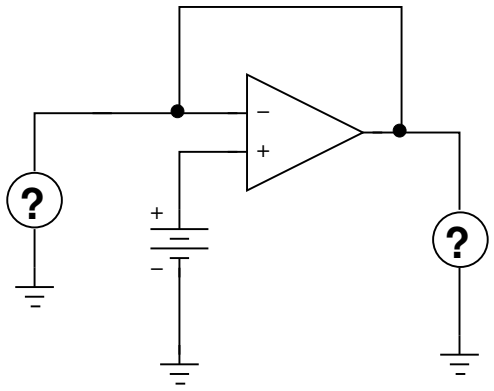

Figure 6.39: *Amplifier in closed-loop configuration with output free in potential.*

To simulate the amplifier in simulator mode, the module parameters $Temp$, V_{DD}, V_{SS}, L_{M_1}, L_{M_2}, L_{M_3}, L_{M_4}, L_{M_5}, L_{M_6}, L_{M_7}, L_{M_8}, W_{M_1}, W_{M_2}, W_{M_3}, W_{M_4}, W_{M_5}, W_{M_6}, W_{M_7}, W_{M_8}, $V_{G/D,M_3}$, V_{OUT}, V_{INP}, V_{INN} and I_{REF,M_8} are chosen. The number of fingers are set in the generator code.

In addition, the Newton-Raphson constraint

$$F_{KCL}(V_{G/D,M_3}) = I_{DS,M_5}(V_{G/D,M_3}) - I_{DS,M_1}(V_{G/D,M_3}) - I_{DS,M_2}(V_{G/D,M_3}) = 0.0 \tag{6.33}$$

is added to ensure that the Kirchhoff's Current Law (KCL) is satisfied at the drain node of M_5. This is satisfied by solving for $V_{G/D,M_3}$ as one unknown.

To account for negative feedback, a Newton-Raphson constraint is added:

$$F_{feedback}(V_{OUT}) = V_{G,M_1}(V_{OUT}) - V_{D,M_7}(V_{OUT}) = 0.0 \tag{6.34}$$

where the unknown is the output terminal voltage V_{OUT}.

To account for the systematic output offset, a Newton-Raphson constraint is added:

$$F_{offset}(V_{INN}) = V_{S,M_1}(V_{INN}) - V_{S,M_2}(V_{INN}) = 0.0 \tag{6.35}$$

where the unknown is the negative input terminal voltage V_{INN}.

The resulting amplifier dependency graph is generated as shown in Fig. 6.40. Note that a conflict exists between the sources V_{S,M_1} and V_{S,M_2}. This gives rise to the introduction of an offset node (C12,V_{OFFSET},4). The design plan represented by this graph is depicted as follows:

1. $V_{G/D,M_8}$ is computed in node (C4,$\{V_{G/D,M_8}, V_{G,M_5}, V_{G,M_7}\}$,31) using the operator $OPVGD(V_{S,M_8}, W_{M_8})$. Note that $V_{G/D,M_8} = V_{G,M_5}, = V_{G,M_7}$

2. Using V_{G,M_7}, I_{DS,M_7} is computed in node (C5,I_{DS,M_7},27) using the operator $OPIDS(V_{G,M_7}, V_{S,M_7})$. This is equally propagated to node (C6,I_{DS,M_6},34) which is used to compute $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$ in node (C7, $\{V_{D,M_2}, V_{D,M_4}, V_{G,M_6}\}$,8) using the operator $OPVG(V_{S,M_6}, W_{M_6})$.

3. The Newton-Raphson constraint node (C12,$F_{offset}(V_{INN})$,0) depends on the voltages V_{S,M_1} and V_{S,M_2} which are computed as follows:
   (a) V_{S,M_1} is computed at node (C11,V_{S,M_1},2) using operator $OPVS(V_{G,M_1}, V_{B,M_1}, W_{M_1})$.
   (b) V_{S,M_2} is computed at node (C10,$\{V_{S,M_2}, V_{D,M_5}\}$,3) using operator $OPVS(V_{G,M_2}, V_{B,M_2}, W_{M_2})$.
   (c) The constraint solves for (C8,V_{INN},1) which is the unknown.
4. The Newton-Raphson constraint node (C12,$F_{KCL}(V_{G/D,M_3})$,45) depends on the currents I_{DS,M_5}, I_{DS,M_1} and I_{DS,M_2} which are computed as follows:
   (a) I_{DS,M_5} is computed at node (C11,I_{DS,M_5},37) using the operator $OPIDS(V_{G,M_5}, V_{S,M_5})$.
   (b) I_{DS,M_3} is computed in node (C9,I_{DS,M_3},12) using the operator $OPIDS(V_{G,M_3}, V_{S,M_3})$. It is then propagated to node (C10,I_{DS,M_1},23) with a weight equal to 1.0.
   (c) I_{DS,M_4} is computed in node (C8,I_{DS,M_4},5) using the operator $OPIDS(V_{G,M_4}, V_{S,M_4})$. It is then propagated to node (C9,I_{DS,M_2},17) with a weight equal to 1.0.
   (d) The constraint solves for (C5,$V_{G/D,M_3}$,46) which is the unknown.
5. The Newton-Raphson constraint node (C12,$F_{feedback}(V_{OUT})$,47) depends on the voltages V_{G,M_1} and V_{D,M_7} which are computed as follows:
   (a) V_{G,M_1} is set via the unknown parameter (C8,V_{INN},1).
   (b) V_{D,M_7} is constant and is set via the parameter (C2,V_{OUT},48)
   (c) The constraint solves for (C2,V_{OUT},48) which is the unknown.

The graph is evaluated using the parameter values defines in Table 6.12. These parameter values are extracted from subsection 6.8.1.3.

After evaluating the graph and solving for $V_{G/D,M_3}$, V_{INN} and V_{OUT}, we get the results in Table 6.13. In this table, the computed systematic input offset is computed for the negative input terminal (or the output node) of the amplifier. The simulator mode results agrees to a good precision with the designer mode results in Table 6.6.

6.9 Conclusion

In this chapter, a circuit sizing and biasing methodology for firm intellectual properties is proposed. Firm intellectual properties are abstracted in the form of modules. A module level consists of a hierarchy of devices and lower-level modules. A bottom-up methodology, that supports both operating-point-driven formulation and standard simulator formulation, automatically generates suitable design plans for firm intellectual properties. This is performed while respecting designer's hypotheses. Design plans are represented using rich module dependency graph. The design plan is executed by evaluating the module dependency graph in a top-down approach. The

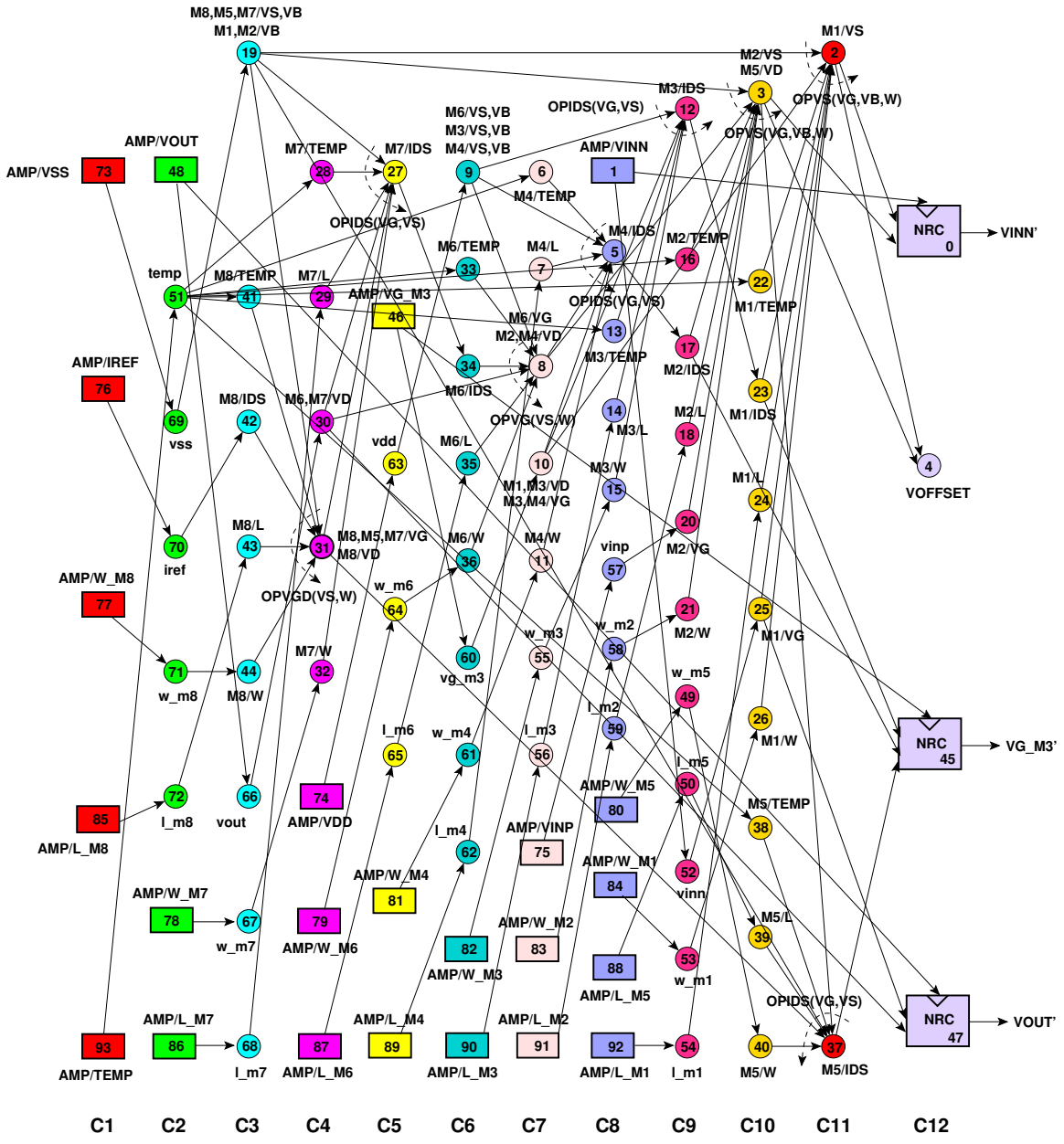

Figure 6.40: *Module dependency graph of the amplifier with systematic output offset and negative feedback in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

methodology ensures knowledge consistency by dealing with under-specified knowledge (such as incomplete knowledge) or over-specified knowledge (such as systematic offset). The methodology also deals with negative feedback circuits. It was successfully illustrated for the synthesis of a

Table 6.12: *Input Parameters For Systematic Output Offset and Negative Feedback in Simulator Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{REF,M_8}(\mu A)$	35.29
$(W/M)_{M_1}(\mu m)$	2.08/4	$L_{M_1}(\mu m)$	0.34
$(W/M)_{M_2}(\mu m)$	2.08/4	$L_{M_2}(\mu m)$	0.34
$(W/M)_{M_3}(\mu m)$	6.32/2	$L_{M_3}(\mu m)$	0.34
$(W/M)_{M_4}(\mu m)$	6.32/2	$L_{M_4}(\mu m)$	0.34
$(W/M)_{M_5}(\mu m)$	6.64/4	$L_{M_5}(\mu m)$	0.34
$(W/M)_{M_6}(\mu m)$	80.63/22	$L_{M_6}(\mu m)$	0.34
$(W/M)_{M_7}(\mu m)$	27.6/16	$L_{M_7}(\mu m)$	0.34
$(W/M)_{M_8}(\mu m)$	6.64/4	$L_{M_8}(\mu m)$	0.34
$V_{G/D,M_3}$(V)	unknown	V_{OUT}(V)	unknown
V_{INN}(V)	unknown	V_{INP}(V)	0.6

Table 6.13: *Computed Parameters for Systematic Output Offset and Negative Feedback in Simulator Mode.*

$V_{G/D,M_3}$	$V_{OUT} = V_{INN}$	$V_{in,off} = V_{INN} - V_{INP}$
0.725942 V	0.600479 V	0.479 mV

two-stage amplifier. The methodology proved to be efficient and accurate in synthesizing generic analog firm intellectual properties.

Chapter 7

Case Studies

7.1 Introduction

To evaluate the effectiveness of our proposed methodology, we apply it for the sizing and biasing of four different analog intellectual properties with various complexity and analog design issues.

In section 7.2, the sizing and biasing of a fully differential cascode current-mode integrator [Smith96] is presented.

In section 7.3, the sizing and biasing of a fully differential common-mode feedback amplifier [Banu88] is presented.

In section 7.4, the sizing and biasing of a fully differential transconductor [Chamla05] is presented.

In section 7.5, the sizing and biasing of a fully differential body-input operational amplifier [Chatterjee05] is presented.

In section 7.6 , we finally conclude the chapter.

7.2 Fully Differential Current-Mode Integrator

The problem of directed cycles caused by incomplete knowledge will be further illustrated on the fully differential cascode current-mode integrator [Smith96] shown in Fig. 7.1. The integrator contains 16 devices shown in dashed boxes. Since all devices in the same row are identically sized and biased, only the first device in each row is sized an biased. In other words, devices M_{888}, M_{666}, M_{444} and M_{222} will be sized and biased. Then, their biases and sizes will be copied into the other devices in their corresponding row. This is done by introducing extrinsic device constraints in the integrator module level. The 15 design parameters of the integrator are $Temp$, V_{DD}, V_{SS}, V_{INCM}, V_{OUTCM}, V_{BIAS}, $V_{eg,M_{222}}$, $V_{eg,M_{444}}$, $V_{eg,M_{666}}$, $V_{eg,M_{888}}$, $L_{M_{222}}$, $L_{M_{444}}$, $L_{M_{666}}$, $L_{M_{888}}$ and I_{BIAS}. Note that V_{CP} and V_{BC} are not given initially. When the *SYNTHESIZE* routine is executed for the integrator module, two directed cycles are detected. The first directed cycle originates from device M_{666}. This directed cycle is shown in bold in Fig. 7.2. The figure shows that $V_{G,M_{666}}$ (labelled as

V_{CP} in Fig. 7.1) and $V_{S,M_{666}}$ are two unknowns that depend on each other. Therefore, this cyclic dependency gives rise to the directed cycle. Again, the same problem appears in device M_{444} as shown in bold in Fig. 7.3. A cyclic dependency between $V_{G,M_{444}}$ (labelled as V_{BC} in Fig. 7.1) and $V_{S,M_{444}}$ gives rise to another directed cycle.

Both device dependency graphs of M_{444} and M_{666} are merged being part of the overall dependency graph of the integrator. The resulting graph contains the same two directed cycles. Therefore, the graph is not a DAG. The two directed cycles are shown in bold in Fig. 7.4. The two directed cycles were retained since they were not resolved. In this case, a dialogue window is displayed to the designer, asking him to resolve both directed cycles. Therefore, the designer should select one parameter from the directed cycle $\{V_{G,M_{666}} = V_{CP}, V_{S,M_{666}}\}$ and another one from $\{V_{G,M_{444}} = V_{BC}, V_{S,M_{444}}\}$. These possible suggestions agree with the choices made in [Aboushady01]. In [Aboushady01], some dependency equations have been introduced in the integrator module to estimate suitable values for V_{CP} and $V_{S,M_{444}}$. Once the directed cycles are resolved, the merging succeeds and the integrator dependency graph is successfully constructed as shown in Fig. 7.5.

The integrator dependency graph is divided into five computational levels. Notice that the fifth computational level shows that all the devices in the same row have the same widths. These were propagated using module constraints. Notice also that the dependency equations introduced by the designer appear at nodes (C2,$V_{CP}(\cdots), 38$) and (C3,$V_{S,M_{444}}(\cdots), 32$).

The integrator graph was evaluated for the parameter values given in Table 7.1. This results in computing the parameters at the last column of the graph. These are given in Table 7.2 for verification. Table 7.3 shows the biases and the small signal parameters for M_{444} and M_{666} in $0.13\mu m$ technology. These are compared to the DC operating point computed by an analog simulator. From the table, it is clear that the proposed methodology is capable of accurately sizing and biasing the integrator.

Table 7.1: *Input Parameters for the Integrator in Designer Mode*.

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.0
V_{SS}(V)	0.0	$I_{BIAS}(\mu$A)	10.0
V_{eg,M_1}(V)	0.22	$L_{M_1}(\mu$m)	7
V_{eg,M_3}(V)	0.1	$L_{M_3}(\mu$m)	5
V_{eg,M_5}(V)	-0.1	$L_{M_5}(\mu$m)	5
V_{eg,M_7}(V)	-0.16	$L_{M_7}(\mu$m)	7
V_{INCM}(V)	0.5	V_{OUTCM}(V)	0.5
V_{BIAS}(V)	0.5		

Table 7.2: *Computed Parameters for the Integrator in Designer Mode.*

$(W/M)_{M_{888}}$	$(W/M)_{M_{666}}$	$(W/M)_{M_{444}}$	$(W/M)_{M_{222}}$	$V_{G,M_{444}}$
67.56$\mu m/2$	105.6$\mu m/3$	27.165$\mu m/1$	8.84$\mu m/2$	0.74665 V

Table 7.3: *Results in 0.13μm technology with $V_{DD} = 1.2V$ for the integrator.*

	CAIRO+		Simulation	
Parameter	M_{444}	M_{666}	M_{444}	M_{666}
$I_{DS}(\mu A)$	10.0	-10.0	10.0	-10.0
$V_{GS}(V)$	0.435081	-0.441122	0.43508	-0.44112
$V_{DS}(V)$	0.188431	-0.253032	0.18849	-0.25297
$V_{BS}(V)$	-0.311569	0.0	-0.31157	0.0
$V_{th}(V)$	0.335081	-0.341122	0.33508	-0.34112
$V_{eg}(V)$	0.1	-0.1	0.1	-0.1
$V_{dsat}(V)$	0.103219	-0.107179	0.10322	-0.10718
$g_m(mA/V)$	0.150713	0.145678	0.15072	0.14568
$g_{ds}(\mu A/V)$	3.81164	0.609771	3.8070	0.61024
$g_{mb}(mA/V)$	0.0285239	0.0353451	0.028524	0.035345
$C_{gd}(fF)$	23.0918	64.748	23.082	64.765
$C_{gs}(pF)$	0.989024	3.52396	0.98901	3.5235
$C_{sd}(fF)$	7.52865	15.6389	7.5207	15.652
$C_{bd}(fF)$	5.60069	10.5656	5.5948	10.574

7.3 Fully Differential Common-Mode Feedback Amplifier

The output balancing behavior of the fully differential common-mode feedback amplifier [Banu88] shown in Fig. 7.6 will be illustrated. The amplifier consists of 8 devices shown in dashed boxes. The amplifier possesses the balancing input voltage V_{BAL}. This input voltage compensates the lack of symmetry that exists in the differential pair (M_{6C}, M_{6A}, M_{6B}) of the common-mode circuit. The differential pair (M_{6C}, M_{6A}, M_{6B}) will not be synthesized separately. Instead, it will be sized and biased identically to the differential pair (M_{1A}, M_{1B}) of the differential-mode input circuit by introducing extrinsic device constraints.

Let us first synthesize the amplifier in the designer mode, the design parameters of the amplifier are chosen: $TEMP$, V_{DD}, V_{SS}, L_{M_5,M_8}, $L_{M_{4B},M_{4A}}$, $L_{M_{3B},M_{3A}}$, $L_{M_{1B},M_{1A}}$, $L_{M_{1BC},M_{1AC}}$, $L_{M_7,M_{2B},M_{2A}}$, $V_{eg,M_{3B},M_{3A}}$, $V_{eg,M_{1B},M_{1A}}$, $V_{eg,M_{1BC},M_{1AC}}$, V_{eg,M_5,M_8}, $V_{eg,M_{2B},M_{2A}}$, V_{D,M_1}, V_{OUTCM}, V_{INCM}, I_{BIAS} and K. Their values are shown in Table 7.4.

After synthesizing the amplifier in the designer mode, the amplifier dependency graph is ob-

Table 7.4: *Input Parameters for the Amplifier in Designer Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{BIAS}(\mu$A)	10.0
V_{INCM}(V)	0.5	V_{OUTCM}(V)	0.5
V_{eg,M_5,M_8}(V)	-0.12	$L_{M_5,M_8}(\mu$m)	0.3
$V_{eg,M_{3B},M_{3A}}$(V)	0.1	$L_{M_{3B},M_{3A}}(\mu$m)	0.3
$V_{eg,M_{1B},M_{1A}}$(V)	-0.12	$L_{M_{1B},M_{1A}}(\mu$m)	0.3
$V_{eg,M_{1BC},M_{1AC}}$,(V)	-0.12	$L_{M_{1BC},M_{1AC}}(\mu$m)	0.3
$V_{eg,M_{2B},M_{2A}}$(V)	0.12	$L_{M_7,M_{2B},M_{2A}}(\mu$m)	0.3
V_{D,M_1}(V)	0.7	$L_{M_{4B},M_{4A}}(\mu$m)	0.3
$K=\frac{I_{M_{4B}}}{I_{M_5}}$	5.0		

tained as shown in Fig. 7.7. The design plan represented by this graph is described as follows:

1. $V_{S,M1B}$ is computed using the operator $OPVS(V_{eg,M_{1B}}, V_{B,M_{1B}})$ in node (C6,$\{V_{S,M1B}, V_{D,M_5}\}$,54).

2. Then, V_{D,M_5} is used to compute V_{G,M_5} and W_{M_5} using operator $OPVG(V_{eg,M_5})$ in nodes (C7,$\{V_{G,M_5}, V_{G,M_{4B}}\}$,44) and (C8,$\{W_{M_5}, W_{M_8}\}$,63) respectively.

3. Then, $V_{G,M_{3B}}$ and $W_{M_{3B}}$ are computed using operator $OPVG(V_{eg,M_{3B}})$ in nodes (C6,$\{V_{G,M_{3B}}, V_{D,M_{1BC}}, V_{D,M_{2B}}\}$,15) and (C8,$\{W_{M_{3A}}, W_{M_{3B}}\}$,35) respectively.

4. Node (C6,$\{V_{G,M_{3B}}, V_{D,M_{1BC}}, V_{D,M_{2B}}\}$,15) is used to compute the biasing voltage $V_{G,M_{1BC}}$ and the width $W_{M_{1BC}}$ using the operator $OPVG(V_{eg,M_{1BC}}, V_{B,M_{1BC}})$ in nodes (C8,$V_{G,M_{1BC}}$,26) and (C8,$\{W_{M_{1BC}}, W_{M_{1AC}}\}$,22) respectively.

5. $V_{G,M_{2B}}$ and $W_{M_{2B}}$ are computed using the operator $OPVG(V_{eg,M_{2B}})$ in nodes (C7,$\{V_{G,M_{2B}}, V_{G/D,M_7}\}$,3) and (C8,$\{W_{M_{2B}}, W_{M_{2A}}\}$,14) respectively.

6. Node (C7,$\{V_{G,M_{2B}}, V_{G/D,M_7}\}$,3) is used to compute W_{M_7} using the operator $OPW(V_{G,M_7}, V_{S,M_7})$ in node (C8,W_{M_7},0).

7. The remaining widths are computed in nodes (C8,$\{W_{M_{4B}}, W_{M_{4A}}\}$,43) and (C8,$\{W_{M_{1B}}, W_{M_{1A}}\}$,51).

Let us study now the output balancing behavior of the amplifier in the simulator mode. Since the widths were computed from the designer mode, those will be used as input parameters for the simulator mode. Table 7.5 shows the values of widths computed in the designer mode.

Table 7.5: *Input Parameters for the Amplifier in Simulator Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	V_{BAL}(V)	unknown
$(W/M)_{M_5}(\mu m)$	3.96/12	$L_{M_5}(\mu m)$	0.3
$(W/M)_{M_8}(\mu m)$	3.96/12	$L_{M_8}(\mu m)$	0.3
$(W/M)_{M_{4A}}(\mu m)$	17.6/40	$L_{M_{4A}}(\mu m)$	0.3
$(W/M)_{M_{4B}}(\mu m)$	17.6/40	$L_{M_{4B}}(\mu m)$	0.3
$(W/M)_{M_{3A}}(\mu m)$	8.12/8	$L_{M_{3A}}(\mu m)$	0.3
$(W/M)_{M_{3B}}(\mu m)$	8.12/8	$L_{M_{3B}}(\mu m)$	0.3
$(W/M)_{M_{1A}}(\mu m)$	1.905/1	$L_{M_{1A}}(\mu m)$	0.3
$(W/M)_{M_{1B}}(\mu m)$	1.905/1	$L_{M_{1B}}(\mu m)$	0.3
$(W/M)_{M_{6C}}(\mu m)$	1.9/2	$L_{M_{6C}}(\mu m)$	0.3
$(W/M)_{M_{6A}}(\mu m)$	0.95/2	$L_{M_{6A}}(\mu m)$	0.3
$(W/M)_{M_{6B}}(\mu m)$	0.95/2	$L_{M_{6B}}(\mu m)$	0.3
$(W/M)_{M_7}(\mu m)$	0.56/1	$L_{M_7}(\mu m)$	0.3
$(W/M)_{M_{2A}}(\mu m)$	0.94/1	$L_{M_{2A}}(\mu m)$	0.3
$(W/M)_{M_{2B}}(\mu m)$	0.94/1	$L_{M_{2B}}(\mu m)$	0.3
$(W/M)_{M_{1AC}}(\mu m)$	2.905/7	$L_{M_{1AC}}(\mu m)$	0.3
$(W/M)_{M_{1BC}}(\mu m)$	2.905/7	$L_{M_{1BC}}(\mu m)$	0.3
V_{OUTM}(V)	unknown	V_{OUTP}(V)	0.5
V_{B_1}(V)	0.724257	V_{B_2}(V)	0.143221
V_{INP}(V)	0.5	V_{INM}(V)	0.5
$V_{G/D,M_7}$(V)	unknown	$V_{S,M_{1A}}$(V)	unknown

The first attempt to simulate the amplifier in the simulator mode will be performed with both inputs and one output fixed to their common-mode voltage levels. The main objective is to compute the output balancing level V_{BAL} for correct common-mode input and output voltages. Therefore, Newton-Raphson constraints are setup for the amplifier as follows:

- To solve for the Kirchhoff's current law at the drain of M_5, we choose to solve for $V_{S,M_{1A}}$ in:

$$F_{KCL,1}(V_{S,M_{1A}}) = I_{DS,M_5}(V_{S,M_{1A}}) - I_{DS,M_{1A}}(V_{S,M_{1A}}) - I_{DS,M_{1B}}(V_{S,M_{1A}}) = 0 \tag{7.1}$$

- To solve for the Kirchhoff's current law at the drain of M_{1A}, we choose to solve for V_{OUTM} in:

$$F_{KCL,2}(V_{OUTM}) = I_{DS,M_{1AC}}(V_{OUTM}) - I_{DS,M_{1A}}(V_{OUTM}) - I_{DS,M_{6A}}(V_{OUTM}) = 0 \tag{7.2}$$

- To solve for the Kirchhoff's current law at the drain of M_{1B}, we choose to solve for V_{BAL} in:

$$F_{KCL,3}(V_{BAL}) = I_{DS,M_{1BC}}(V_{BAL}) - I_{DS,M_{1B}}(V_{BAL}) - I_{DS,M_{6B}}(V_{BAL}) = 0 \tag{7.3}$$

- To solve for the Kirchhoff's current law at the drain of M_8, we choose to solve for $V_{G/D,M_7}$ in:

$$F_{KCL,4}(V_{G/D,M_7}) = I_{DS,M_8}(V_{G/D,M_7}) - I_{DS,M_{6C}}(V_{G/D,M_7}) - I_{DS,M_{6A}}(V_{G/D,M_7}) - I_{DS,M_{6B}}(V_{G/D,M_7}) = 0 \tag{7.4}$$

Therefore Table 7.5 contains the four unknowns $V_{S,M_{1A}}$, V_{OUTM}, V_{BAL} and $V_{G/D,M_7}$. The negative feedback constraint is treated differently by adding a designer-defined procedure that computes the gate voltage of M_{6C} as $\frac{V_{OUTP}+V_{OUTM}}{2}$. This procedure accounts for the functionality of the resistive networks connected at the differential output.

After simulating the amplifier in simulator mode, the amplifier module dependency graph is obtained as shown in Fig. 7.8. The design plan represented by the dependency graph simply computes the currents for all transistors and then solves the Newton-Raphson constraints in the nodes of column C8.

Evaluating the dependency graph using the values in Table 7.5, the four unknowns are computed as shown in Table 7.6. Note that V_{OUTP} is set at its correct common-mode level. In this case, the systematic offset in the output balancing level V_{BAL} is 3.267mV.

Table 7.6: *Computed Parameters for the Amplifier in Simulator Mode with* $V_{OUTP} = 0.5V$.

$V_{S,M_{1A}}$	V_{OUTP}	V_{BAL}	$V_{G/D,M_7}$
1.01062 V	0.5 V	0.496733 V	0.476428 V

Now, we see what happens to the common-mode output level if the inputs and the balancing level are kept at their correct common-mode levels. To examine what happens, the design plan is kept the same, with the exception of changing the Newton-Raphson constraints as follows:

- To solve for the Kirchhoff's current law at the drain of M_5, we choose to solve for $V_{S,M_{1A}}$ in:

$$F_{KCL,1}(V_{S,M_{1A}}) = I_{DS,M_5}(V_{S,M_{1A}}) - I_{DS,M_{1A}}(V_{S,M_{1A}}) - I_{DS,M_{1B}}(V_{S,M_{1A}}) = 0 \tag{7.5}$$

- To solve for the Kirchhoff's current law at the drain of M_{1A}, we choose to solve for V_{OUTM} in:

$$F_{KCL,2}(V_{OUTM}) = I_{DS,M_{1AC}}(V_{OUTM}) - I_{DS,M_{1A}}(V_{OUTM}) - I_{DS,M_{6A}}(V_{OUTM}) = 0 \tag{7.6}$$

- To solve for the Kirchhoff's current law at the drain of M_{1B}, we choose to solve for V_{OUTP} in:

$$F_{KCL,3}(V_{OUTP}) = I_{DS,M_{1BC}}(V_{OUTP}) - I_{DS,M_{1B}}(V_{OUTP}) - I_{DS,M_{6B}}(V_{OUTP}) = 0 \tag{7.7}$$

- To solve for the Kirchhoff's current law at the drain of M_8, we choose to solve for $V_{G/D,M_7}$ in:

$$F_{KCL,4}(V_{G/D,M_7}) = I_{DS,M_8}(V_{G/D,M_7}) - I_{DS,M_{6C}}(V_{G/D,M_7}) - I_{DS,M_{6A}}(V_{G/D,M_7}) - I_{DS,M_{6B}}(V_{G/D,M_7}) = 0 \tag{7.8}$$

Table 7.7: *Computed Parameters for Amplifier in Simulator Mode with* $V_{BAL} = 0.5V$.

$V_{S,M_{1A}}$	V_{OUTP}	V_{OUTM}	$V_{G/D,M_7}$
1.01062 V	0.503095 V	0.503095 V	0.476358 V

For this purpose, the input parameters are kept the same as in Table 7.5, except that $V_{BAL} = 0.5V$ and V_{OUTP} becomes an unknown. After synthesizing the amplifier and evaluating the graph, we get the value of the four new unknowns as in Table 7.7

It is clear that the output common-mode level has moved to $\frac{0.503095V+0.503095V}{2} = 0.503095V$ with an output systematic offset of 3.095mV. Note that this value is equal to the gate voltage of M_{6C} which is the other branch of the differential-input of the common-mode circuit. Hence, it is expected to have an offset value similar to the one found from Table 7.6. The difference comes from the lack of symmetry between the differential-mode inputs of the common-mode circuit.

Now what happens if the output offset is brought to the differential-mode inputs of the amplifier circuit. To examine this case, the amplifier outputs and the balancing level voltage are kept at their correct common-mode voltage levels. Only, the inputs are allowed to change. To perform this, the Newton-Raphson constraints were setup as follows:

- To solve for the Kirchhoff's current law at the drain of M_5, we choose to solve for $V_{S,M_{1A}}$ in:

$$F_{KCL,1}(V_{S,M_{1A}}) = I_{DS,M_5}(V_{S,M_{1A}}) - I_{DS,M_{1A}}(V_{S,M_{1A}}) - I_{DS,M_{1B}}(V_{S,M_{1A}}) = 0 \tag{7.9}$$

- To solve for the Kirchhoff's current law at the drain of M_{1A}, we choose to solve for V_{INM} in:

$$F_{KCL,2}(V_{INM}) = I_{DS,M_{1AC}}(V_{INM}) - I_{DS,M_{1A}}(V_{INM}) - I_{DS,M_{6A}}(V_{INM}) = 0 \tag{7.10}$$

- To solve for the Kirchhoff's current law at the drain of M_{1B}, we choose to solve for V_{INP} in:

$$F_{KCL,3}(V_{INP}) = I_{DS,M_{1BC}}(V_{INP}) - I_{DS,M_{1B}}(V_{INP}) - I_{DS,M_{6B}}(V_{INP}) = 0 \tag{7.11}$$

- To solve for the Kirchhoff's current law at the drain of M_8, we choose to solve for $V_{G/D,M_7}$ in:

$$F_{KCL,4}(V_{G/D,M_7}) = I_{DS,M_8}(V_{G/D,M_7}) - I_{DS,M_{6C}}(V_{G/D,M_7}) - I_{DS,M_{6A}}(V_{G/D,M_7}) - I_{DS,M_{6B}}(V_{G/D,M_7}) = 0 \tag{7.12}$$

For this purpose, the input parameters are kept the same as in Table 7.5, except that $V_{BAL} = V_{OUTM} = V_{OUTP} = 0.5$. Also both V_{INM} and V_{INP} become unknowns. After evaluating the amplifier graph, we get the value of the four new unknowns as in Table 7.8.

From the table, the common-mode input voltage is equal to $0.396834V$ which accounts for a systematic input offset 103.166mV. Comparing this with the offset obtained in Table 7.6, we conclude that the differential gain of the common-mode circuit is 33 times higher than the common-mode gain of differential-mode of the amplifier.

Table 7.8: *Computed Parameters for Amplifier in Simulator Mode with* $V_{OUTP} = V_{OUTM} = 0.5V$.

$V_{S,M_{1A}}$	V_{INP}	V_{INM}	$V_{G/D,M_7}$
0.927481 V	0.396834 V	0.396834 V	0.478045 V

7.4 Fully Differential Transconductor

The complex hierarchy of the fully differential transconductor [Chamla05] shown Fig. 7.9 will be explored. The transconductor consists of four subcircuits as shown in Fig. J.1:

1. One instance of the differential transconductance subcircuit (GMD).
2. Two instances of the amplifier feedback subcircuit (AMP).
3. One instance of the common-mode feedback circuit (CMC).

The transconductor will be synthesized as follows: each subcircuit will be synthesized as a standalone subcircuit. This step will result in one dependency graph for each subcircuit. To obtain the dependency graph for the whole transconductor, the dependency graphs of the four subcircuits will be merged using the method of equipotentials described in section 6.4.1.

Let us examine in more details how each subcircuit is synthesized in the designer mode. For convenience, the graphs are given in appendix J. The dependency graph of GMD subcircuit is obtained as shown in Fig. J.2. The dependency graph of CMC subcircuit is obtained as shown in Fig. J.3. Moreover, the dependency graph of AMP subcircuit is obtained as shown in Fig. J.4. For convenience, the figures related to this section have been placed in appendix J. After merge , the dependency graph of the transconductor, its module dependency graphs contains 375 nodes.

The hierarchical interconnection between subcircuits is shown in Fig. J.1. The parameters shown in this figure are key parameters since they allow to construct each subcircuit separately and then combine the four subcircuits to construct the transconductor. Table 7.9 shows the input parameters for each subcircuit. The set of parameters of the transconductor is the addition of all the subcircuit parameters. Fig. J.1 names some connectors that becomes internal to the transconductor instead of being external for standalone subcircuits. For example, connector (GMD/VCMFB) becomes internal to the transconductor. Therefore, its corresponding parameter disappears from the final parameter list of the transconductor.

The transconductor dependency graph is then evaluated. The resulting dimensions and biases are then used to simulate the transconductor in the simulator mode.

In the simulator mode, we would like to plot the effective transconductance of the transconductor which is defined as follows:

$$GM_{eff} = \frac{\partial(\Delta I_{DS,OUTP} - \Delta I_{DS,OUTN})}{\partial(V_{IN+} - V_{IN-})} \tag{7.13}$$

$$= \frac{\partial\{(I_{DS,M9AN} - I_{DS,M2N}) - (I_{DS,M9AP} - I_{DS,M2P})\}}{\partial(V_{IN+} - V_{IN-})} \tag{7.14}$$

Table 7.9: *Input Parameters for the Transconductor in Designer Mode.*

Differential Transconductance Subcircuit (GMD)			
Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	V_{INCM}(V)	0.8
V^1_{OUTCM}(V)	0.8	V_C(V)	0.4
V^1_{CMFB}(V)	0.8	$I_{DS,M_{1AP}}(\mu$A)	40.0
$V_{eg,M_{2P}}$(V)	0.05	$L_{M_{1AP}}(\mu$m)	3.0
$L_{M_{2P}}(\mu$m)	1.0	$L_{M_{9AP}}(\mu$m)	1.0
Amplifier Feedback Subcircuit (AMP)			
Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	V_C(V)	0.4
$I_{C,BIAS}(\mu$A)	12.0	$I_{S,BIAS}(\mu$A)	-6.0
$V_{eg,M_{4BP}}$(V)	-0.1	$L_{M_{4BP}}(\mu$m)	3.0
$V_{eg,M_{8BP}}$(V)	-0.1	$L_{M_{8BP}}(\mu$m)	1.0
$V_{eg,M_{3BP}}$(V)	0.1	$L_{M_{3BP}}(\mu$m)	3.0
$V_{eg,M_{7BP}}$(V)	-0.1	$L_{M_{7BP}}(\mu$m)	1.0
$V_{eg,M_{5BP}}$(V)	-0.1	$L_{M_{5BP}}(\mu$m)	1.0
$V_{eg,M_{10AP}}$(V)	0.1	$L_{M_{10AP}}(\mu$m)	3.0
$V^2_{D,M_{3BP}}$(V)	0.84	$L_{M_{6BP}}(\mu$m)	1.0
Common-Mode Feedback Subcircuit (CMC)			
Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	$I_{BIAS}(\mu$A)	12
$V_{eg,M_{11AP}}$(V)	0.1	$L_{M_{11AP}}(\mu$m)	1
$V_{eg,M_{12AP}}$(V)	0.1	$L_{M_{12AP}}(\mu$m)	3
$V_{eg,M_{13}}$(V)	-0.1	$L_{M_{13}}(\mu$m)	1
V_{REF}(V)	0.8	V^3_{CMFB}(V)	0.8

1,2,3 disappears in the final list of the full transconductor

For this purpose, we need to perform a DC sweep analysis by sweeping control voltage V_C while drawing GM_{eff} versus the difference between the common-mode voltages at the two input terminals of GMD. The input parameters for the transconductor have been selected as shown in Table7.10.

To simulate the full transconductor in simulator mode, Newton-Raphson constraints have been set in each subcircuit. Then the dependency graph of the four subcircuits are merged to pro-

Table 7.10: *Input Parameters for the Transconductor in Simulator Mode.*

Differential Transconductance Subcircuit (GMD)				Common-Mode Feedback Subcircuit (CMC)			
Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2	$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	V_{CMFB}(V)	0.8	V_{SS}(V)	0.0	V_{CMFB}(V)	0.8
$V_{S,M_{2N}}$(V)	0.4	$V_{S,M_{2P}}$(V)	0.4	$V_{S,M_{11AN}}$(V)	0.415	$V_{S,M_{11AP}}$(V)	0.415
V_{INN}(V)	0.8	V_{INP}(V)	0.8	$(W/M)_{M_{13}}(\mu$m)	24.64/77	$L_{M_{13}}(\mu$m)	1.0
$V_{G,M_{2N}}$(V)	0.842	$V_{G,M_{2P}}$(V)	0.842	$(W/M)_{M_{11AP}}(\mu$m)	3.4/8	$L_{M_{11AP}}(\mu$m)	1.0
$(W/M)_{M_{1AN}}(\mu$m)	3.445/1	$L_{M_{1AN}}(\mu$m)	3.0	$(W/M)_{M_{11BP}}(\mu$m)	3.4/8	$L_{M_{11BP}}(\mu$m)	1.0
$(W/M)_{M_{1AP}}(\mu$m)	3.445/1	$L_{M_{1AP}}(\mu$m)	3.0	$(W/M)_{M_{11AN}}(\mu$m)	3.4/8	$L_{M_{11AN}}(\mu$m)	1.0
$(W/M)_{M_{2N}}(\mu$m)	50.44/8	$L_{M_{2N}}(\mu$m)	1.0	$(W/M)_{M_{11BN}}(\mu$m)	3.4/8	$L_{M_{11BN}}(\mu$m)	1.0
$(W/M)_{M_{2P}}(\mu$m)	50.44/8	$L_{M_{2P}}(\mu$m)	1.0	$(W/M)_{M_{12AP}}(\mu$m)	21.59/17	$L_{M_{12AP}}(\mu$m)	3.0
$(W/M)_{M_{9AN}}(\mu$m)	145.46/28	$L_{M_{9AN}}(\mu$m)	1.0	$(W/M)_{M_{12AN}}(\mu$m)	21.59/17	$L_{M_{12AN}}(\mu$m)	3.0
$(W/M)_{M_{9AP}}(\mu$m)	145.46/28	$L_{M_{9AP}}(\mu$m)	1.0				
V_{OUT}(V)	0.8						

Amplifier Feedback Subcircuit (AMP)			
Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.2
V_{SS}(V)	0.0	V_C(V)	0.4
$V_{S,M_{3BP}}$(V)	0.439	$V_{D,M_{3BP}}$(V)	0.842
$V_{S,M_{4AP}}$(V)	0.887	$V_{S,M_{4BP}}$(V)	0.887
$V_{G,M_{4AP}}$(V)	0.4	$V_{G,M_{10AP}}$(V)	0.363
$V_{G,M_{5BP}}$(V)	0.763	$V_{G,M_{7BP}}$(V)	0.577
$(W/M)_{M_{5BP}}(\mu$m)	12.92/8	$L_{M_{5BP}}(\mu$m)	1.0
$(W/M)_{M_{5AP}}(\mu$m)	12.92/8	$L_{M_{5AP}}(\mu$m)	1.0
$(W/M)_{M_{6BP}}(\mu$m)	50.4/32	$L_{M_{6BP}}(\mu$m)	1.0
$(W/M)_{M_{6AP}}(\mu$m)	50.4/32	$L_{M_{6AP}}(\mu$m)	1.0
$(W/M)_{M_{7AP}}(\mu$m)	12.92/19	$L_{M_{7AP}}(\mu$m)	1.0
$(W/M)_{M_{7BP}}(\mu$m)	12.92/19	$L_{M_{7BP}}(\mu$m)	1.0
$(W/M)_{M_{8BP}}(\mu$m)	12.73/19	$L_{M_{8BP}}(\mu$m)	1.0
$(W/M)_{M_{8AP}}(\mu$m)	12.73/19	$L_{M_{8AP}}(\mu$m)	1.0
$(W/M)_{M_{4BP}}(\mu$m)	39.76/16	$L_{M_{4BP}}(\mu$m)	3.0
$(W/M)_{M_{4AP}}(\mu$m)	39.76/16	$L_{M_{4AP}}(\mu$m)	3.0
$(W/M)_{M_{3BP}}(\mu$m)	10.56/16	$L_{M_{3BP}}(\mu$m)	3.0
$(W/M)_{M_{3AP}}(\mu$m)	10.56/16	$L_{M_{3AP}}(\mu$m)	3.0
$(W/M)_{M_{10AP}}(\mu$m)	21.44/32	$L_{M_{10AP}}(\mu$m)	3.0

duce the transconductor dependency graph. In this case, the transconductor dependency graph inherits the Newton-Raphson constraints of all subcircuits. This way, knowledge reuse for firm IP is capitalized.

For a complete description, we give the Newton-Raphson constraints for each subcircuit separately.

For the GMD subcircuit, the dependency graph shown in Fig. J.5 contains the following Newton-Raphson constraints:

- To solve for the Kirchhoff's current law at the source of M_{2P}, we choose to solve for $V_{S,M_{2P}}$ in:

$$F_{KCL,1}(V_{S,M_{2P}}) = I_{DS,M_{2P}}(V_{S,M_{2P}}) - I_{DS,M_{1AP}}(V_{S,M_{2P}}) = 0 \tag{7.15}$$

- To solve for the Kirchhoff's current law at the source of M_{2N}, we choose to solve for $V_{S,M_{2N}}$ in:

$$F_{KCL,2}(V_{S,M_{2N}}) = I_{DS,M_{2N}}(V_{S,M_{2N}}) - I_{DS,M_{1AN}}(V_{S,M_{2N}}) = 0 \tag{7.16}$$

- To compute the GM_{eff} in the simulator mode, we connect V_{OUT+} and V_{OUT-} with a short circuit forming one node V_{OUT}. Since the output current of both branches are equal, we choose to solve for V_{OUT} in:

$$F_{KCL,3}(V_{OUT}) = -I_{DS,M_{9AN}}(V_{OUT}) - I_{DS,M_{2N}}(V_{OUT}) - I_{DS,M_{2P}}(V_{OUT}) - I_{DS,M_{9AP}}(V_{OUT}) = 0 \tag{7.17}$$

For the CMC subcircuit, the dependency graph shown in Fig. J.6 contains the following Newton-Raphson constraints:

- To solve for the Kirchhoff's current law at the source of M_{11AP}, we choose to solve for $V_{S,M_{11AP}}$ in:

$$F_{KCL,1}(V_{S,M_{11AP}}) = I_{DS,M_{12AP}}(V_{S,M_{11AP}}) - I_{DS,M_{11AP}}(V_{S,M_{11AP}}) - I_{DS,M_{11BP}}(V_{S,M_{11AP}}) = 0 \tag{7.18}$$

- To solve for the Kirchhoff's current law at the source of M_{11AN}, we choose to solve for $V_{S,M_{11AN}}$ in:

$$F_{KCL,2}(V_{S,M_{11AN}}) = I_{DS,M_{12AN}}(V_{S,M_{11AN}}) - I_{DS,M_{11AN}}(V_{S,M_{11AN}}) - I_{DS,M_{11BN}}(V_{S,M_{11AN}}) = 0 \tag{7.19}$$

- To solve for the Kirchhoff's current law at the drain of M_{13}, we choose to solve for V_{CMFB} in:

$$F_{KCL,3}(V_{CMFB}) = -I_{DS,M_{13}}(V_{CMFB}) - I_{DS,M_{11AN}}(V_{CMFB}) - I_{DS,M_{11BP}}(V_{CMFB}) = 0 \tag{7.20}$$

For the AMP subcircuit, the dependency graph shown in Fig. J.7 contains the following Newton-Raphson constraints:

- To solve for the Kirchhoff's current law at the source of M_{4BP}, we choose to solve for $V_{S,M_{4BP}}$ in:

$$F_{KCL,1}(V_{S,M_{4BP}}) = I_{DS,M_{5BP}}(V_{S,M_{4BP}}) - I_{DS,M_{4BP}}(V_{S,M_{4BP}}) = 0 \tag{7.21}$$

- To solve for the Kirchhoff's current law at the source of M_{4AP}, we choose to solve for $V_{S,M_{4AP}}$ in:

$$F_{KCL,2}(V_{S,M_{4AP}}) = I_{DS,M_{5AP}}(V_{S,M_{4AP}}) - I_{DS,M_{4AP}}(V_{S,M_{4AP}}) = 0 \tag{7.22}$$

- To solve for the Kirchhoff's current law at the drain of M_{3BP}, we choose to solve for $V_{D,M_{3BP}}$ in:

$$F_{KCL,3}(V_{D,M_{3BP}}) = I_{DS,M_{3BP}}(V_{D,M_{3BP}}) + I_{DS,M_{6BP}}(V_{D,M_{3BP}}) = 0 \tag{7.23}$$

- To solve for the Kirchhoff's current law at the drain of M_{3AP}, we choose to solve for $V_{D,M_{3AP}}$ in:

$$F_{KCL,4}(V_{D,M_{3AP}}) = I_{DS,M_{3AP}}(V_{D,M_{3AP}}) + I_{DS,M_{6AP}}(V_{D,M_{3AP}}) = 0 \tag{7.24}$$

- To solve for the Kirchhoff's current law at the source of M_{3AP}, we choose to solve for $V_{S,M_{3AP}}$ in:

$$F_{KCL,5}(V_{S,M_{3AP}}) = I_{DS,M_{10AP}}(V_{S,M_{3AP}}) - I_{DS,M_{3BP}}(V_{S,M_{3AP}}) - I_{DS,M_{3AP}}(V_{S,M_{3AP}}) = 0 \tag{7.25}$$

It is important to note that the final transconductor dependency graph contains 598 nodes and 11 Newton-Raphson constraints. To simulate the transconductor in simulator mode, a procedure is written to sweep the control voltage V_C. At each sweep point, the graph is evaluated by changing the input common-mode voltages in opposite directions.

After simulation in the simulator mode, the computed effective transconductance is compared against a simulation using an analog simulator. The plot in Fig. 7.10 shows that our proposed methodology achieves results that are identical to commercial simulators. Slid lines ar issued from an analog simulation and points are issued from our proposed methodology. Moreover, the resulting linearity of the transconductor fully agrees with the results obtained in the work related to [Chamla05] for 0.13μm CMOS technology with $V_{DD} = 1.2V$.

7.5 0.5V Power Supply Fully Differential Body-Input Operational Amplifier

In this section, we present the fully differential body-input operational amplifier [Chatterjee05] used for very low supply voltages. This circuit has be chosen since it has uncommon circuit topology and it operates under very low voltage. The amplifier is shown in Fig. 7.11. It consists of 7 transistor devices. The resistances R_A and R_B will be accounted for using a designer-defined procedure that computes the drain voltage of M_4 using R_A and I_{DS,M_4}.

The design parameters for the amplifier are chosen to be $TEMP$, V_{DD}, V_{SS}, V_{OUTCM}, V_{INCM}, $L_{M_{1A},M_{1B}}$, $L_{M_{2A},M_{2B}}$, $L_{M_{3A},M_{3B}}$, L_{M_4}, $I_{DS,M_4} = I_{BIASI}$, $I_{DS,M_{2A},M_{2B}} = I_{BIASN}$, $V_{eg,M_{2A},M_{2B}}$, V_{eg,M_4}, K and $R_{A,B}$. The parameter values are given in Table 7.11.

After synthesizing the amplifier, the design plan is represented by the dependency graph shown in Fig. 7.12. The computed parameters of the amplifier are given in Table 7.12.

Table 7.11: *Input Parameters For Body-Input Amplifier in Designer Mode.*

Parameter	Value	Parameter	Value
$TEMP$(Kelvin)	300.15	V_{DD}(V)	1.0
V_{SS}(V)	0.0	V_{OUTCM}(V)	0.25
V_{INCM}(V)	0.25	$R_{A,B}$(KOhms)	100
$I_{BIASN}(\mu$A)	40.0	$I_{BIASI}(\mu$A)	4.0
$K = \frac{I_{DS,M_{1A},M_{1B}}}{I_{DS,M_{3A}},M_{3B}}$	6	$L_{M_4}(\mu$m)	2.0
$V_{eg,M_{2A},M_{2B}}$(V)	0.1	$L_{M_{1A},M_{1B}}(\mu$m)	1.0
V_{eg,M_4}(V)	0.1	$L_{M_{3A},M_{3B}}(\mu$m)	1.0
$L_{M_{2A},M_{2B}}(\mu$m)	1.0		

Table 7.12: *Computed Parameters for The Amplifier.*

$(W/M)_{M_{1A},M_{1B}}$	$(W/M)_{M_{2A},M_{2B}}$	$(W/M)_{M_{3A},M_{3B}}$	$(W/M)_{M_4}$	V_{G,M_4}	$V_{G,M_{2A},M_{2B}}$
$35.94\mu m/3$	$22.03\mu m/2$	$6.375\mu m/3$	$7.25\mu m/1$	0.391658 V	0.4137 V

Synthesis results are compared against the simulation results as shown in Table 7.13. The results show that the proposed methodology can still be used for very low voltage applications of future analog circuits.

Table 7.13: *Operating Point for The amplifier.*

Parameter	Synthesis			Simulation		
	M_{1A}	M_{3A}	M_4	M_{2A}	M_{3A}	M_4
$I_{DS}(\mu A)$	-36.0	-6.0	4.0	-36.004	-6.0036	4.0011
$V_{GS}(V)$	-0.45	-0.45	0.391658	-0.45	-0.45	0.39166
$V_{DS}(V)$	-0.25	-0.25	0.05	-0.24995	-0.24995	0.05
$V_{BS}(V)$	-0.25	-0.25	0.0	-0.24997	-0.24995	0.0
$V_{th}(V)$	-0.29615	-0.296361	0.291658	-0.29615	-0.29637	0.29166
$V_{eg}(V)$	-0.15385	-0.153639	0.1	-0.15385	-0.15363	0.1
$V_{dsat}(V)$	-0.13744	-0.13571	0.1	-0.13744	-01.3571	0.1
$g_m(mA/V)$	0.397796	0.0664513	0.0481461	0.39784	0.066493	0.048159
$g_{ds}(\mu A/V)$	6.51391	1.05594	53.3472	6.5184	1.0572	53.362
$g_{mb}(\mu A/V)$	59.0094	8.90347	10.5272	59.020	8.91	10.530
$C_{gd}(fF)$	16.9596	3.00258	25.2097	16.964	3.0048	25.217
$C_{gs}(pF)$	0.290907	0.0513384	0.100128	0.29093	0.051367	0.10015
(Operating Region)	saturation	saturation	linear	saturation	saturation	linear

7.6 Conclusion

In this chapter, the proposed methodology has been used to synthesize four different analog intellectual properties: *fully differential current-mode integrator*, *fully differential common-mode feedback amplifier*, *fully differential transconductor* and *0.5V power supply fully differential body-input operational amplifier*. The different aspects of analog design discussed in the previous chapter have been demonstrated in each case study. The proposed methodology proved to be efficient and accurate allowing to automatically produce design plans that respect designer's hypotheses.

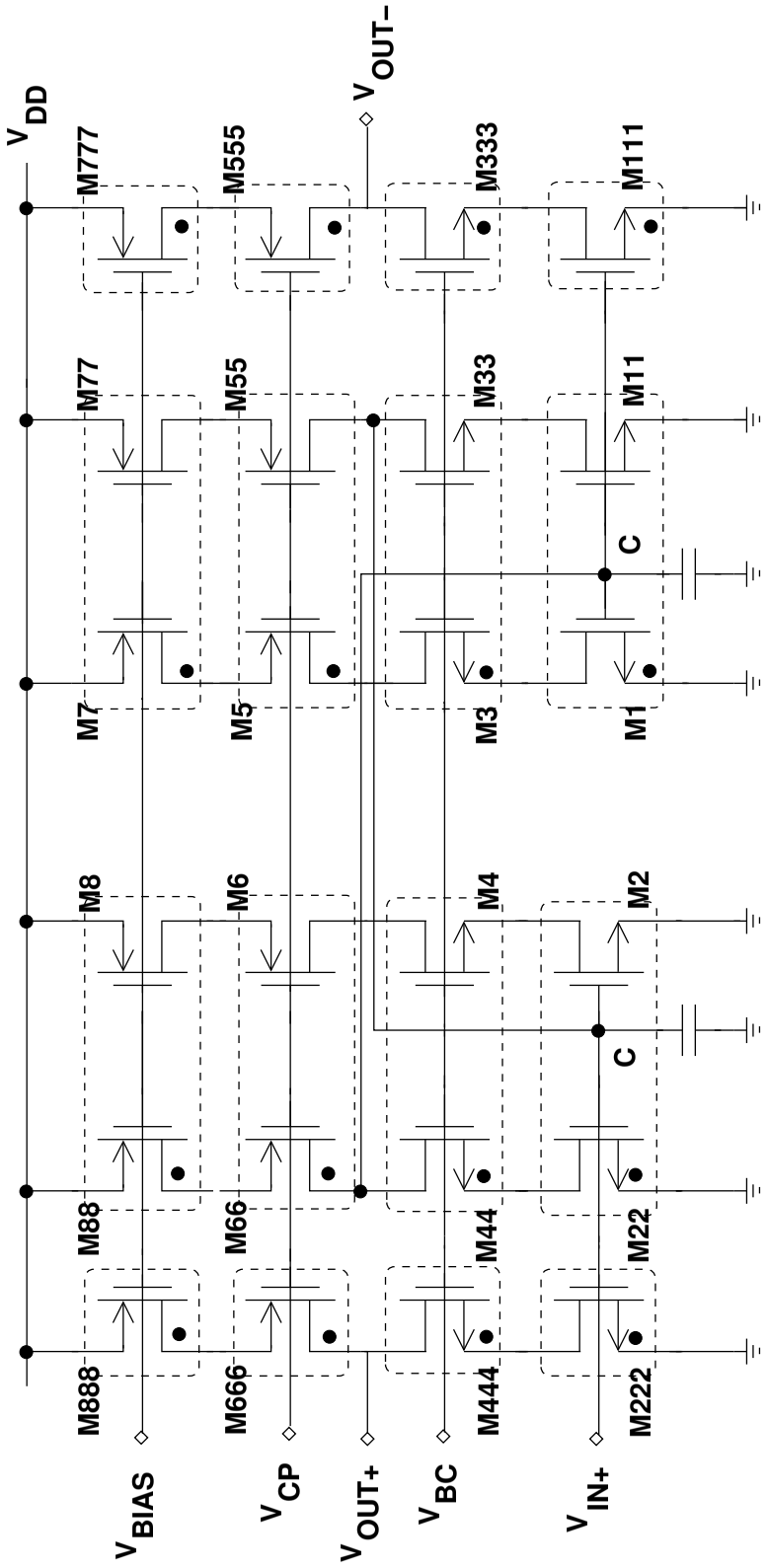

Figure 7.1: *Fully differential current-mode integrator.*

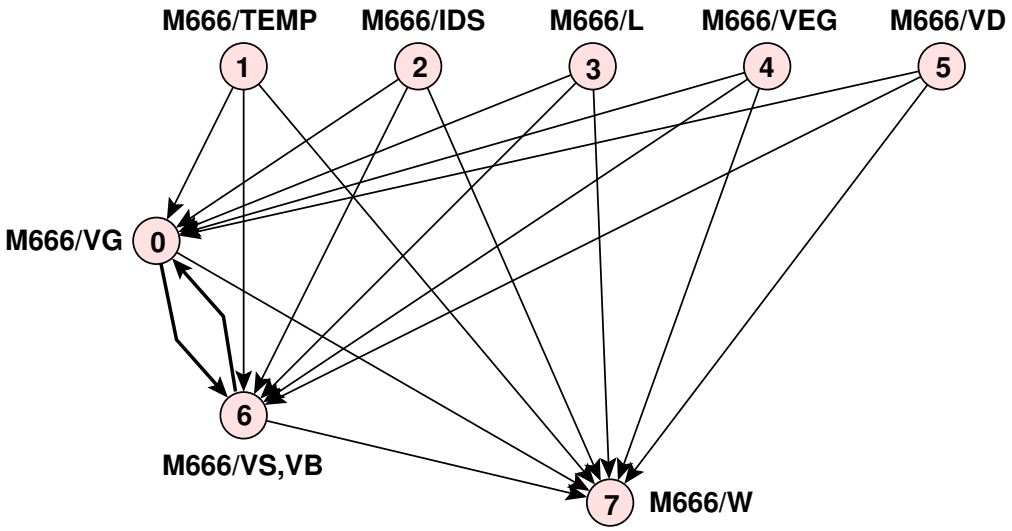

Figure 7.2: *Directed cycle for the transistor* M_{666}.

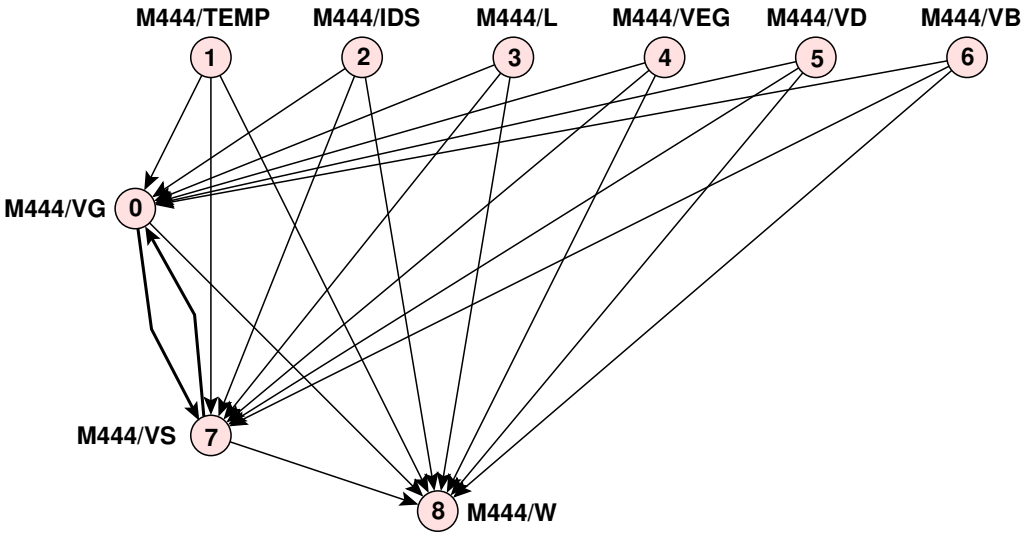

Figure 7.3: *Directed cycle for the transistor* M_{444}.

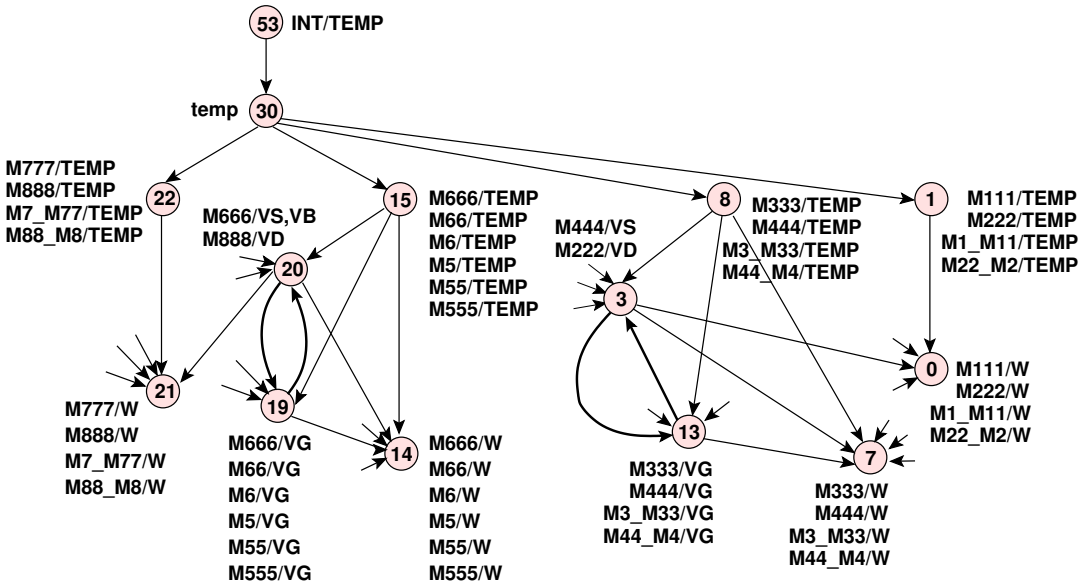

Figure 7.4: *Directed cycles in the integrator dependency graph.*

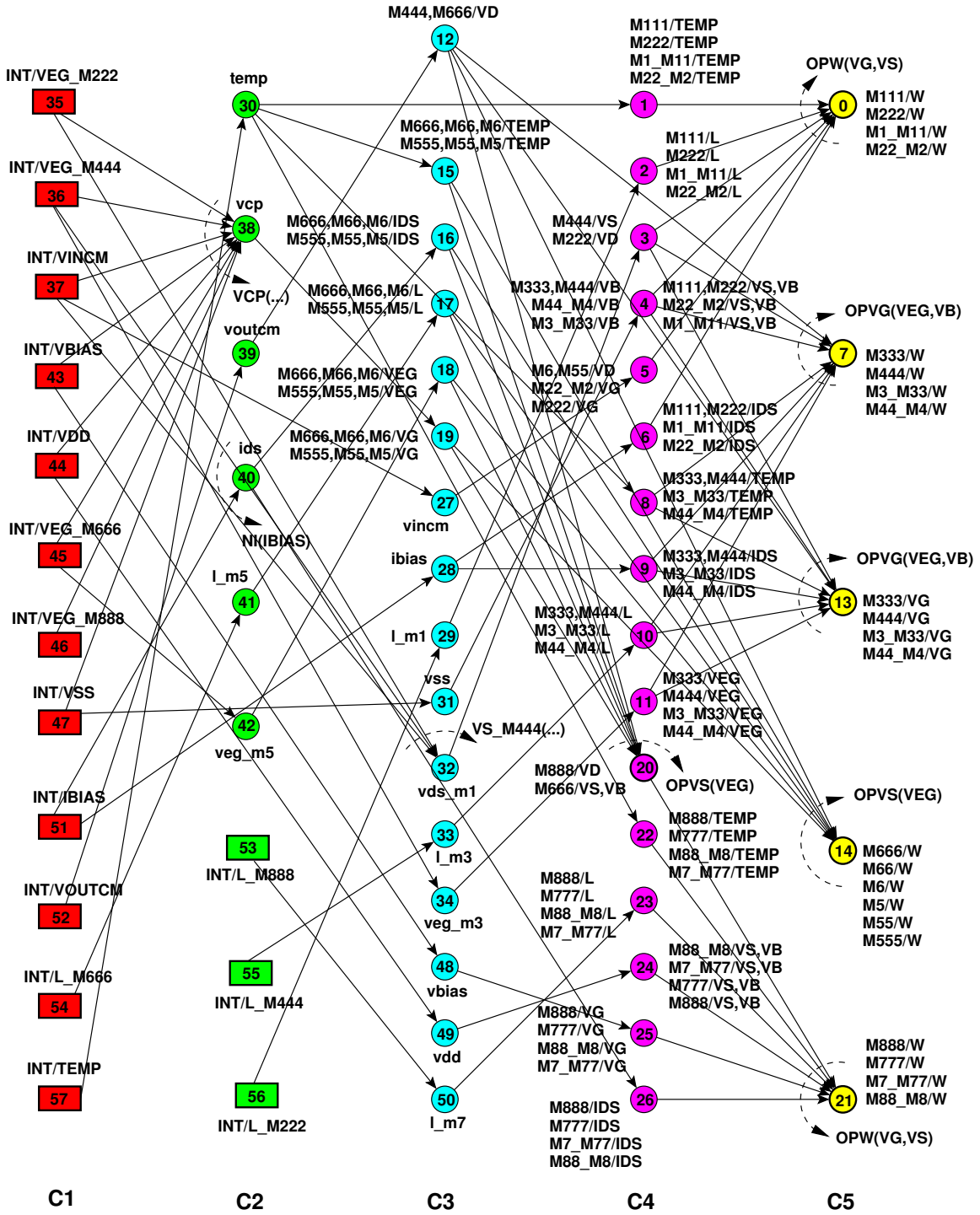

Figure 7.5: *Module dependency graph of the fully differential current-mode integrator in designer mode: (a) Rectangles are integrator parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

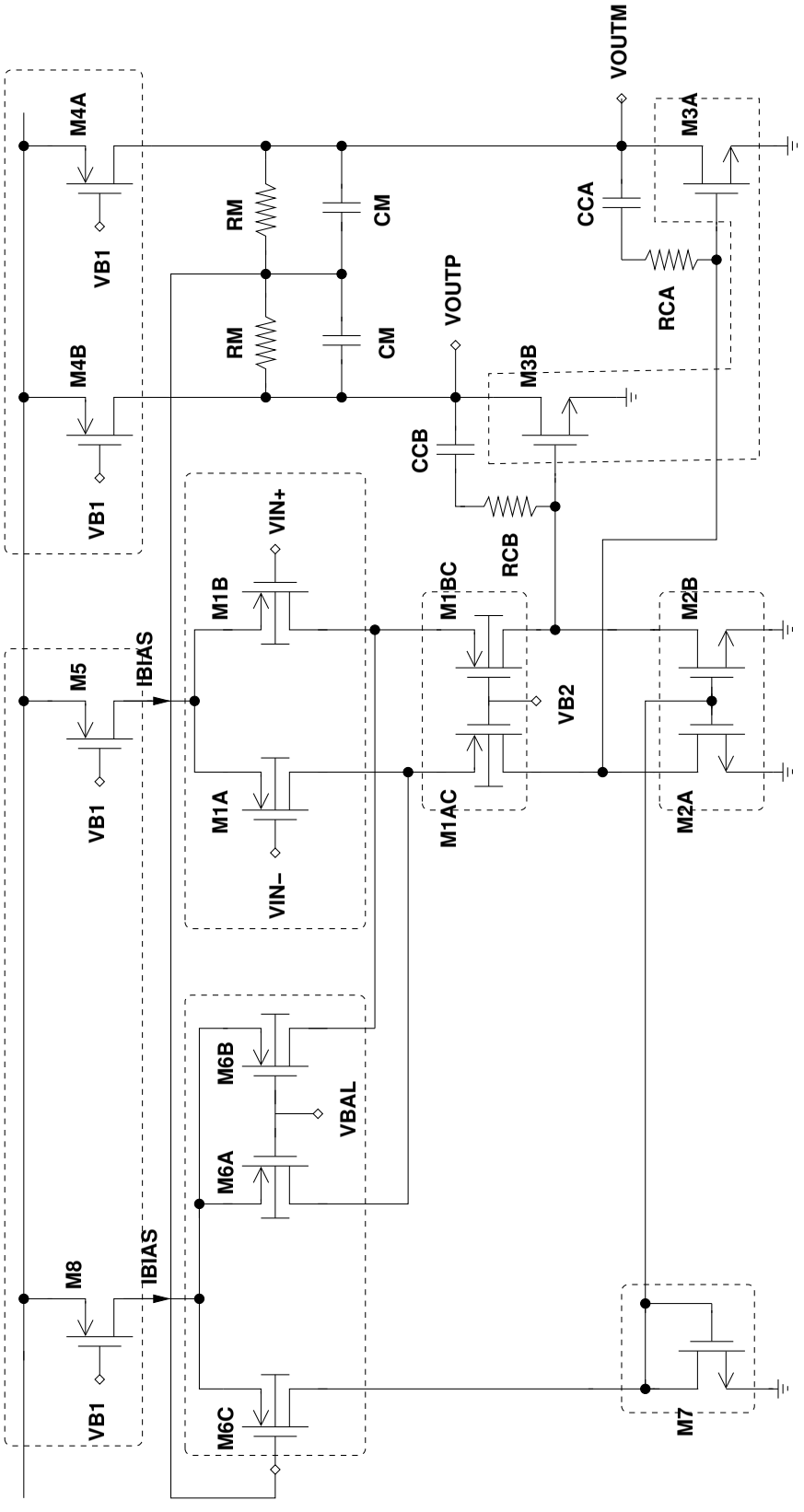

Figure 7.6: *Fully differential common-mode feedback amplifier.*.

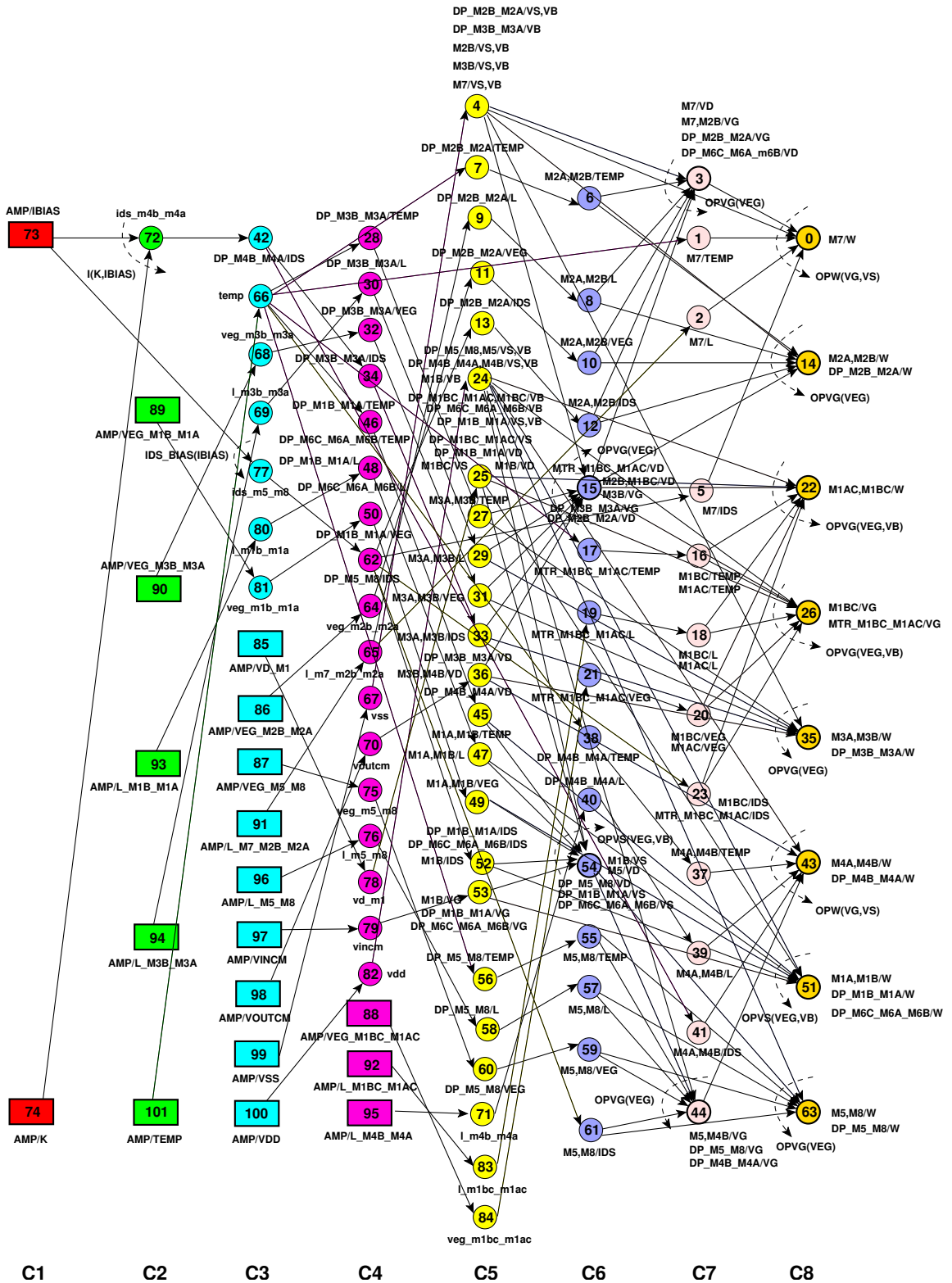

Figure 7.7: *Module dependency graph of the fully differential common-mode feedback amplifier in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

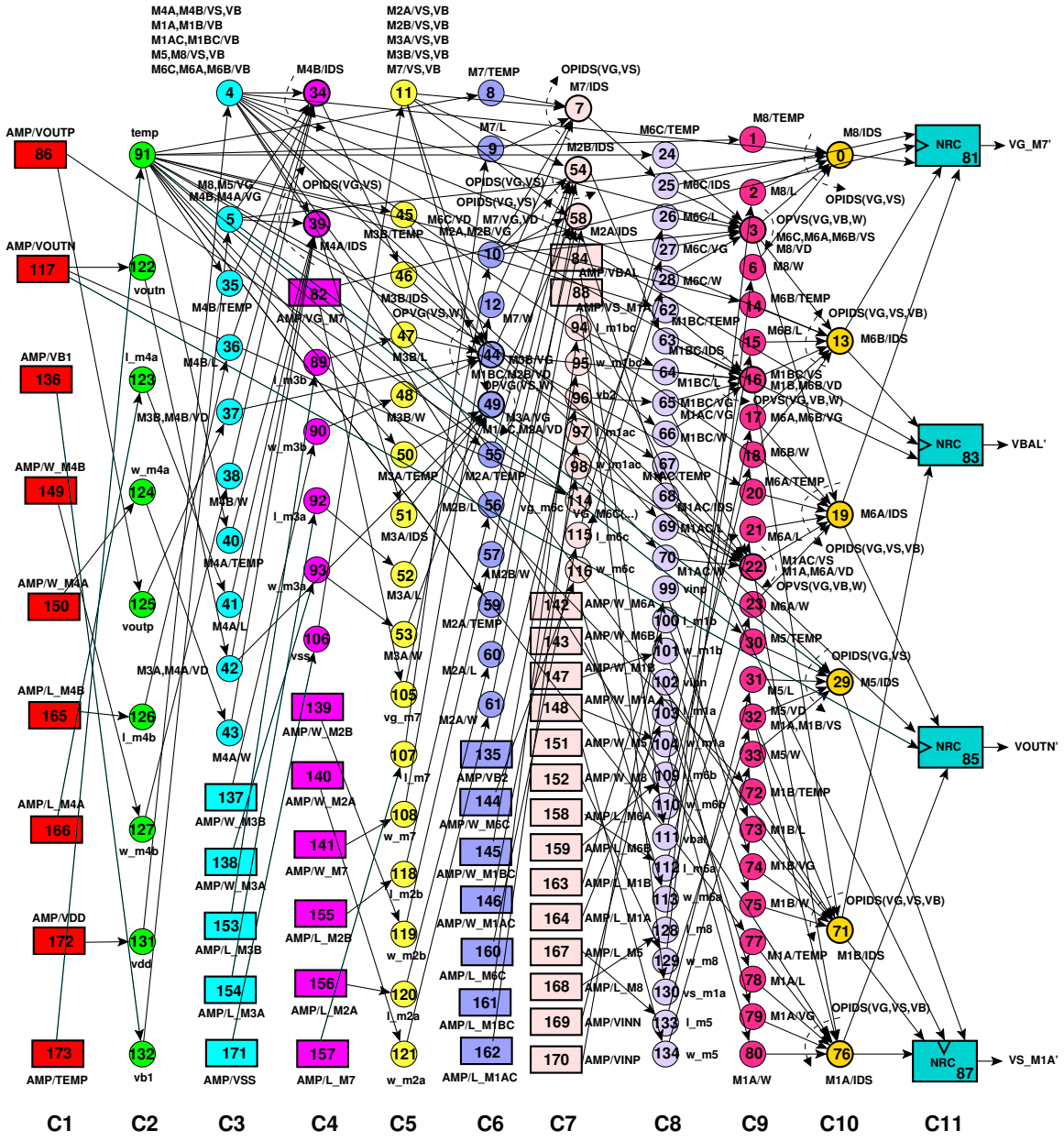

Figure 7.8: *Module dependency graph of the fully differential common-mode feedback amplifier in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

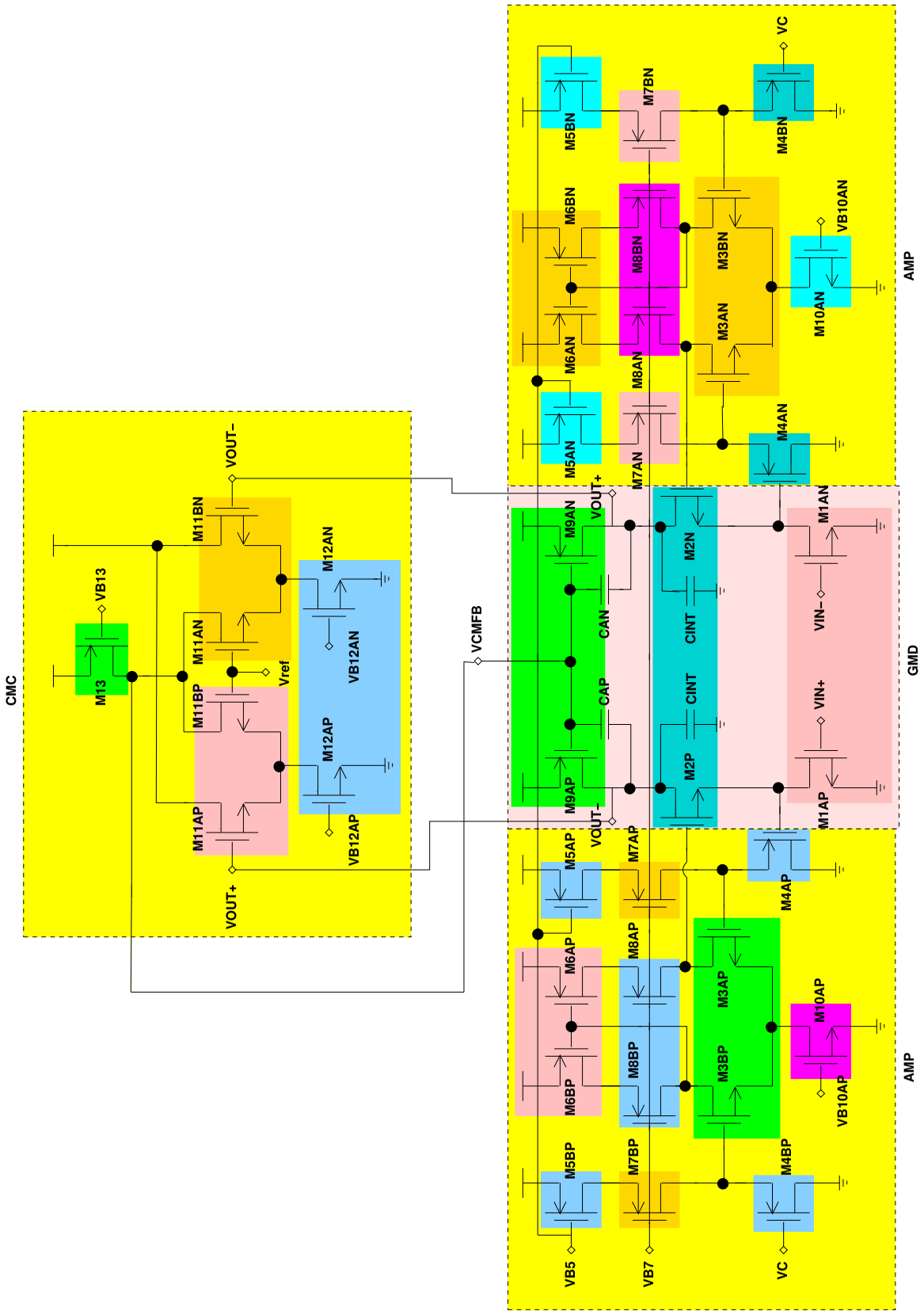

Figure 7.9: *Fully differential transconductor.*

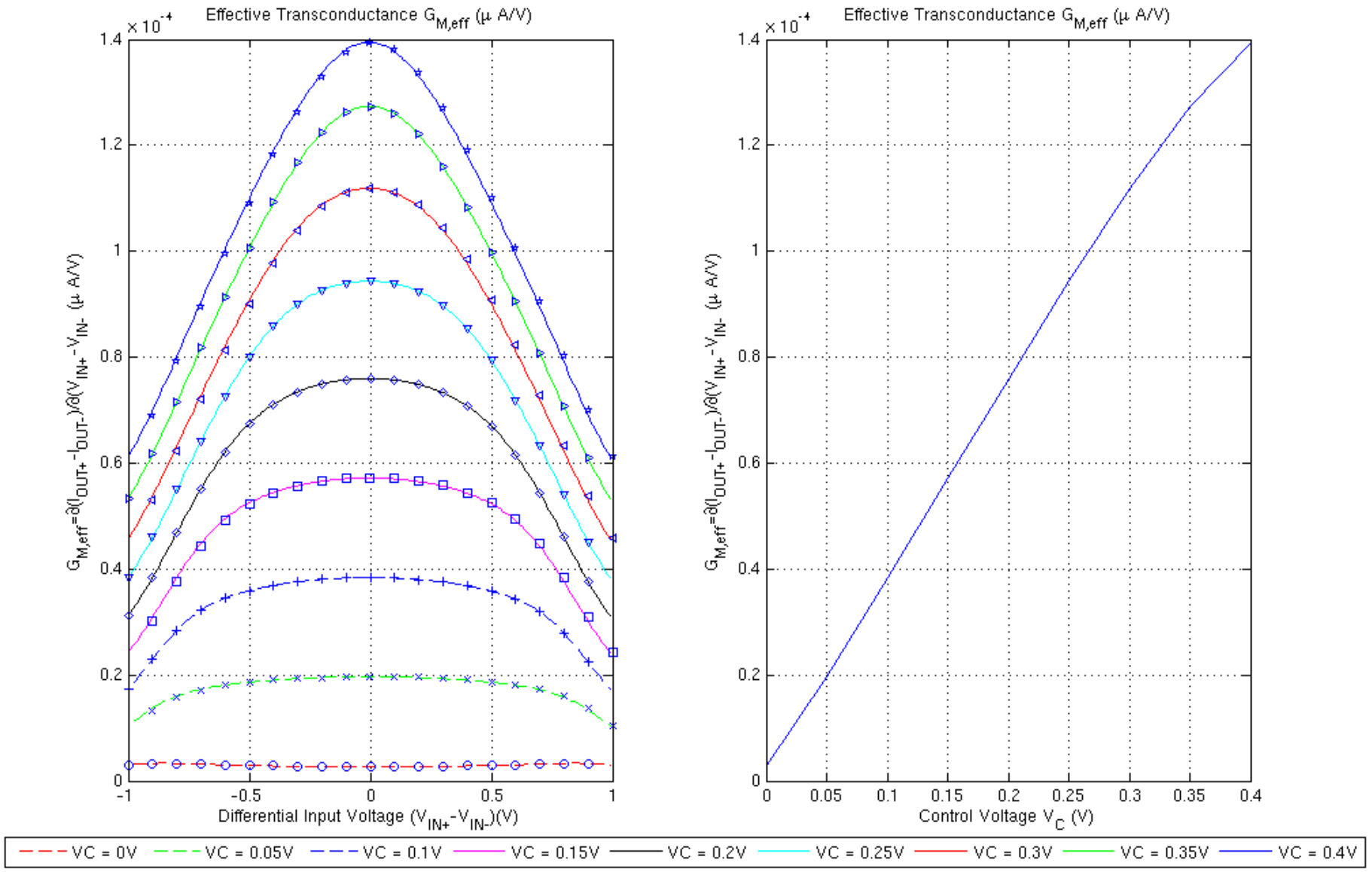

Figure 7.10: *Fully differential transconductor response.*

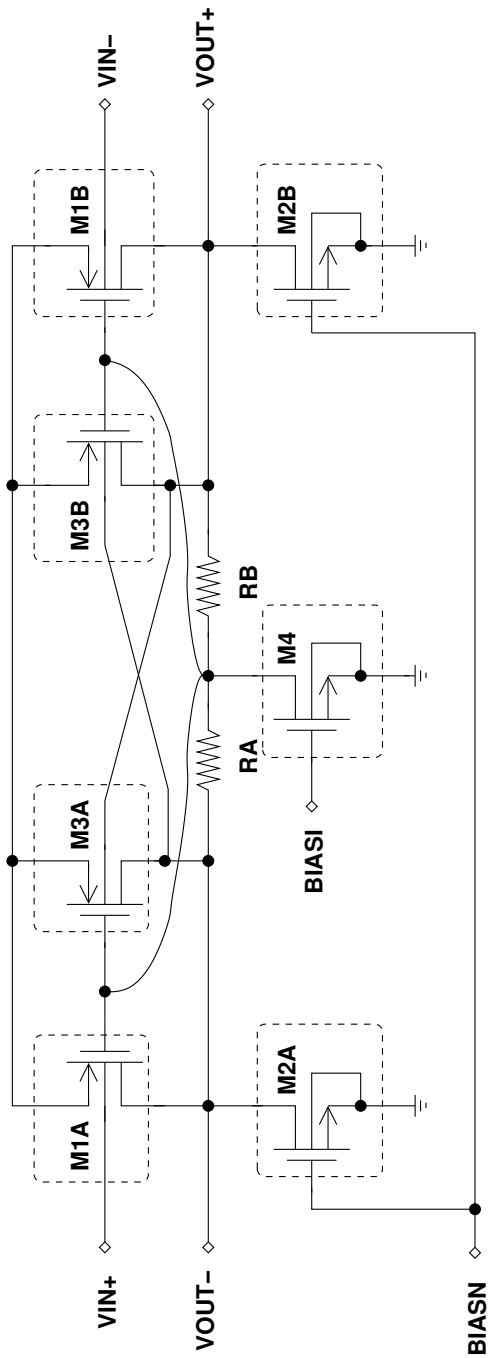

Figure 7.11: *Fully differential body-Input operational amplifier*.

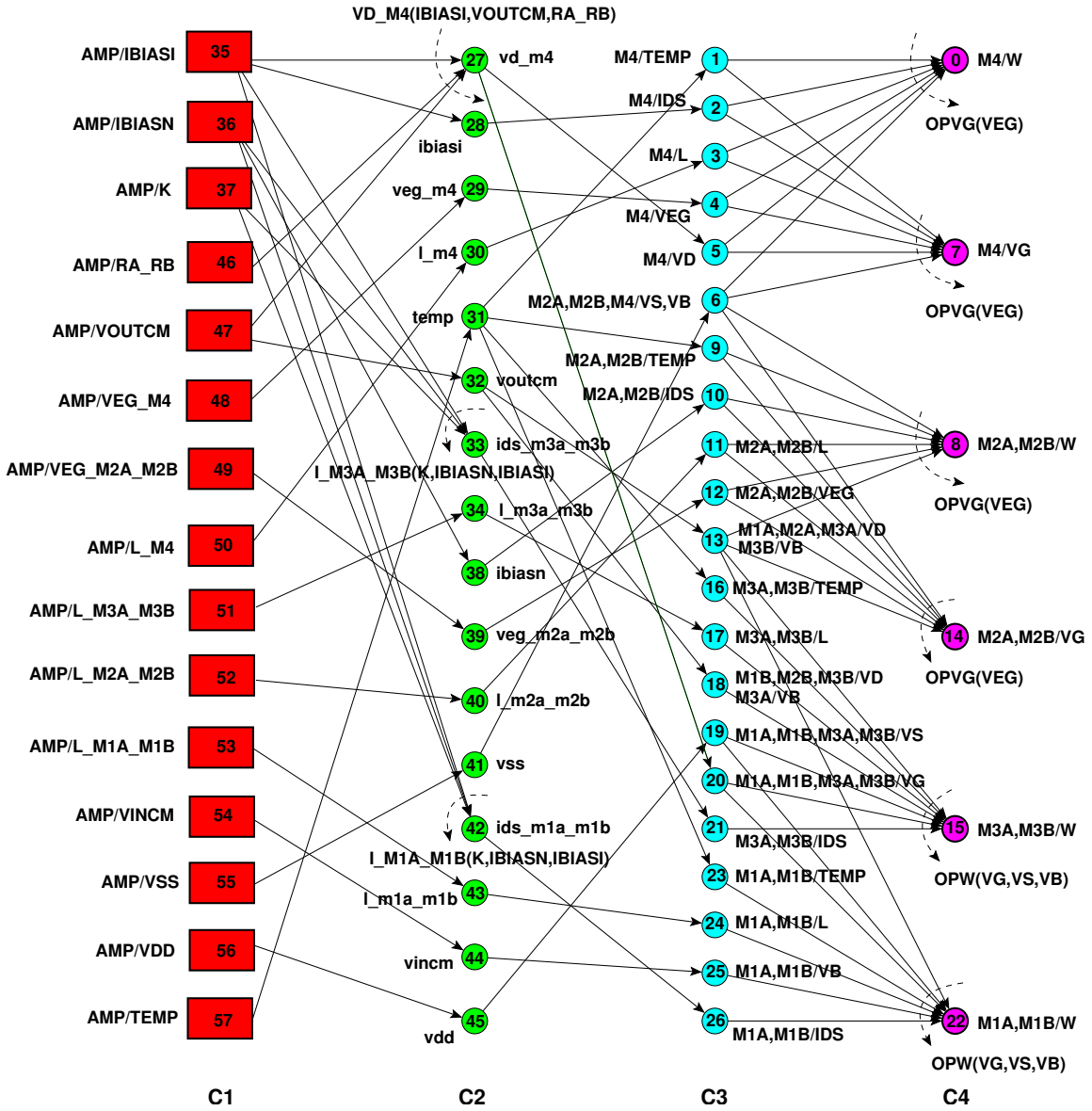

Figure 7.12: *Module dependency graph of the fully differential body-input operational amplifier in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

Chapter 8

Knowledge-Aware Synthesis

8.1 Introduction

The main motivation of this chapter is to prove the benefits of introducing the hierarchical sizing and biasing inside an optimization loop of a knowledge-based synthesis system. Traditionally, simulation-based synthesis systems use transistor widths as their main optimization variables. This is due to the fact that these systems interface with simulators in order to evaluate circuit performances. Inherently, the choice of widths as optimization variables is not optimal. Since the range of widths is large and generally chosen arbitrarily, the design space becomes large. Consequently, synthesis systems spend an important amount of execution time in evaluating infeasible designs. A more convenient set of variables that can be used for optimization consists of voltages, currents and lengths. These variables have a very narrow range: voltages range from V_{SS} to V_{DD}, current ranges are determined from system specifications and lengths are fixed by the designer. Hence, these variables represent a much smaller design space for the same circuit. Moreover, since widths are always computed from these variables, the synthesized designs tend to be feasible.

Section 8.2 describes the concept of the knowledge-aware optimization-based synthesis.

Section 8.3 illustrates some optimization results. An OTA amplifier is first synthesized using our proposed methodology. A comparison with the results of the state-of-art knowledge-based and simulation-based synthesis tools is discussed. Then, the proposed methodology is used to migrate the OTA amplifier by changing one specification.

Finally, section 8.4 draws some conclusions about the effectiveness of the proposed methodology.

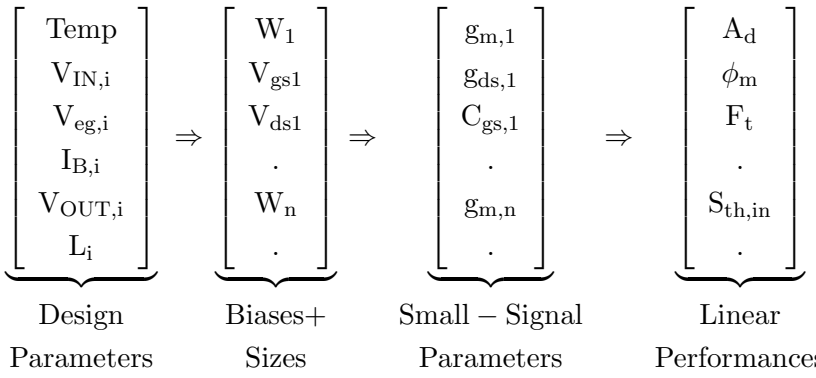

Figure 8.1: *Parameter mappings in the design space..*

8.2 Knowledge-Aware Optimization-Based Synthesis

8.2.1 The Choice of Optimization Variables

The hierarchical sizing and biasing method performs the parameter mappings shown in Fig. 8.1. Since voltages, currents and lengths have a narrow range of variation, they represent a much reduced design space for the same circuit compared to design spaces using widths. Consequently, the first vector is favorably used for optimization. In the second vector, the sizes and biases are computed out of the first vector. Therefore elements of the second vector tend to be feasible compared to the simulation-based synthesis where they are arbitrarily chosen causing some execution time to be lost in evaluating infeasible designs.

8.2.2 The Reduction Factor

To quantify the reduction in the design space due to the selection of appropriate optimization variables, we deduced a simple figure of merit called *the reduction factor* which is defined as the ratio between the number of possible values of a width $W_i = W_i^{min} : W_i^{max} : \lambda_i^w$ and a voltage $V_i = V_i^{min} : V_i^{max} : \lambda_i^v$,

$$Reduction\ Factor = \prod_{i=1}^{n} \frac{\lambda_i^v}{\lambda_i^w} \cdot \frac{W_i^{max} - W_i^{min}}{V_i^{max} - V_i^{min}} \tag{8.1}$$

where n is the number of widths variables to be exchanged by voltage variables, λ_i^w is the step size of the width that varies between W_i^{min} and W_i^{max}, and λ_i^v is the step size of the voltage that varies between V_i^{min} and V_i^{max}. We conclude that exchanging a width with a voltage leads to an important reduction factor in the design space. This is due to the more limited supply voltage as well as the broader choice of widths for shrinking physical grid, accompanying technology advances. This reduction factor is described by the contour shown in Fig. 8.2 for one dimensional problem, i.e. $n = 1$. The figure was produced for 0.13μm CMOS technology with $W_i^{min} = 0.15\mu m$, $W_i^{max} = 100\mu m$, $V_i^{min} = 0V$ and $V_i^{max} = 1.2V$. Around the voltage step size of 1mV and the

width step size of 10nm, the design space is reduced by at least nine times. Applying equation 8.1 for $n > 1$, we get a very important reduction factor for the n-dimensional design space.

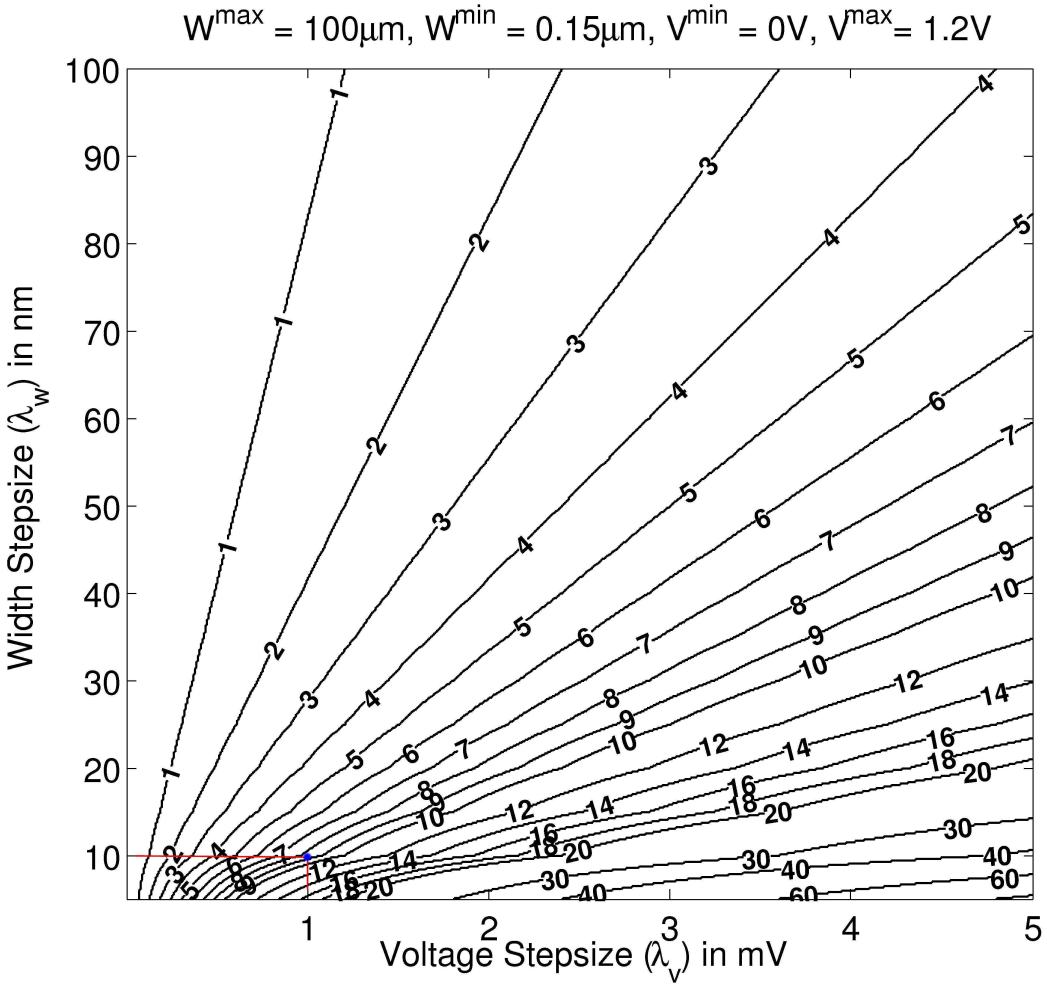

Figure 8.2: *Reduction factor for 0.13μm CMOS technology.*

Some designers may disagree with the *reduction in design space* argument, since it is unfair to compare discrete spaces (defined by widths) to continuous spaces (defined by voltages). On the other hand, one might discretize a voltage by estimating the minimum permissible voltage error in the circuit. Note that the minimum permissible voltage error is much lower than errors due to mismatch. It should be mentioned also that the voltage precision should not affect the functionality of the circuit, otherwise the circuit is not robust.

8.2.3 Optimization Engine

To achieve the potential reduction in the design space during synthesis, the hierarchical sizing and biasing have been introduced into the optimization loop as shown in Fig. 8.3.

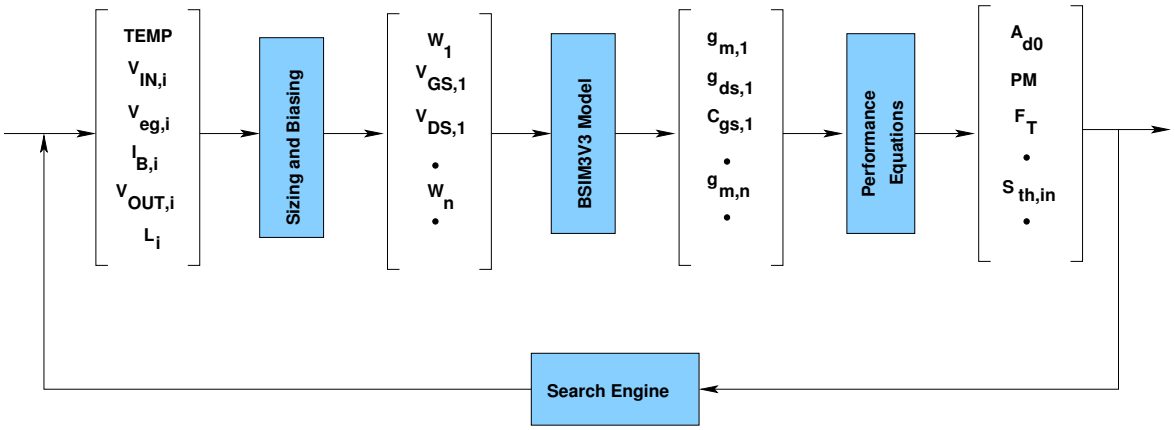

Figure 8.3: *Block diagram of the proposed synthesis system.*

The optimizer generates the elements of the first vector of Fig. 8.1. Next, all widths and biases of the second vector are computed from the first one using the generated design plan. Then, the small-signal parameters of the third vector are computed from the second one by evaluating the BSIM3V3 model equations. The linear performance equations in the fourth vector are evaluated in the root circuit level. These equations are manually coded by the designer inside a procedure called *performance procedure* which is called by the optimizer to evaluate performances. Finally, the performance values are sent back to the optimizer to estimate new parameter values for the first vector. The loop continues until it converges to a solution vector that satisfies all performance constraints.

If a solution becomes infeasible during optimization, a very large value is returned back for the cost function causing the solution to be rejected. This is done immediately after evaluating the module dependency graph during optimization in order to get sizes and biases in the second vector.

The optimization algorithm used is the *Nelder-Mead Simplex* [Nelder65, Lagarias98]. The Nelder-Mead algorithm attempts to minimize a scalar-valued nonlinear cost function of n real variables using only function values, without any derivative information. The algorithm maintains at each step a non-degenerate *simplex*, a geometric figure in n dimensions of nonzero volume that is the convex hull of $n + 1$ vertices. In two dimensions, a simplex is a triangle. In three dimensions it is a tetrahedron, not necessarily regular tetrahedron.

The method starts by defining an initial simplex consisting of $n + 1$ points. If one of these points is the initial starting point $\mathbf{P}_0$, then one can take the other n points to be:

$$\mathbf{P}_i = \mathbf{P}_0 + \lambda \cdot \mathbf{e}_i \tag{8.2}$$

where the $\mathbf{e}_i$'s is the n units vector, and where λ is a constant which is our guess of the problem's characteristic length scale. Or, one could have different λ_i's for each vector direction.

To start the method we need to choose the initial simplex to start. The algorithm starts by ordering the $n+1$ vertices to satisfy:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1}) \tag{8.3}$$

where $f(.)$ is the cost function. The algorithm is then supposed to make its own way downhill through the complex n-dimensional response surface, until it encounters an (at least local) minimum. The Nelder-Mead algorithm takes a series of steps, most steps just moving the point of the simplex where the function is highest $\mathbf{x}_{n+1}$ (the point of simplex where the function is largest) through the opposite face of the simplex to a lower point $\mathbf{x}_r$. This step is called *reflection*, and it is constructed to conserve the volume of the simplex. This is shown in Fig. 8.4(b). The reflection point $\mathbf{x}_r$is computed from:

$$\mathbf{x}_r = \bar{\mathbf{x}} + \rho(\bar{\mathbf{x}} - \mathbf{x}_{n+1}) \tag{8.4}$$

where:

$$\bar{\mathbf{x}} = \sum_{i=1}^{n} \frac{\mathbf{x}_i}{n}$$

is the centroid of the n best points, i.e. all vertices except for $\mathbf{x}_{n+1}$ and ρ is the reflection coefficient. The cost function f_r is evaluated at $\mathbf{x}_r$, $f_r = f(\mathbf{x}_r)$. If $f_1 \leq f_r \leq f_n$, the reflected point $\mathbf{x}_r$ is accepted and the iteration is terminated.

When the value of function in point $\mathbf{x}_r$ is lower than or equal to the lowest point of the simplex, it means that we have better estimation of the minimum. Therefore, the value is checked in point $\mathbf{x}_e$ to see if the function drops further in direction of $\mathbf{x}_r$. This is called *expansion*. It is shown as a further movement in the direction of minimization in Fig.8.4(b). The expansion point $\mathbf{x}_e$ is computed from:

$$\mathbf{x}_e = \bar{\mathbf{x}} + \chi(\mathbf{x}_r - \bar{\mathbf{x}}) \tag{8.5}$$

where χ is the expansion coefficient. The cost function is valuated at $\mathbf{x}_e$, $f_e = f(\mathbf{x}_e)$. If $f_e < f_r$, the expansion point $\mathbf{x}_e$ is accepted, otherwise (if $f_e \geq f_r$), the reflection point $\mathbf{x}_r$ is accepted and the iteration is terminated.

If after a reflection, $\mathbf{x}_r$ is still the worst point, then a simple *contraction* step is made between $\bar{\mathbf{x}}$ and the better of $\mathbf{x}_{n+1}$ and $\mathbf{x}_r$.

1. If $f_n \leq f_r < f_{n+1}$, an *outside contraction* is performed. The outside contraction point $\mathbf{x}_c$ is shown if Fig. 8.4(c). It is computed from

$$\mathbf{x}_c = \bar{\mathbf{x}} + \gamma(\mathbf{x}_r - \bar{\mathbf{x}}) \tag{8.6}$$

   where γ is the contraction coefficient. The function is evaluated $f_c = f(\mathbf{x}_c)$. If $f_c \leq f_r$ the contraction point x_c is accepted and the iteration is terminated.

2. If $f_r \geq f_{n+1}$, perform an *inside contraction*. The inside contraction point x_{cc} is computed from

$$\mathbf{x}_{cc} = \bar{\mathbf{x}} - \gamma(\bar{\mathbf{x}} - \mathbf{x_{n+1}}) \tag{8.7}$$

The function is evaluated $f_{cc} = f(\mathbf{x}_{cc})$. If $f_{cc} < f_{n+1}$, the contraction point x_{cc} is accepted and the iteration is terminated.

If a simple contraction step does not improve the situation, then all the points are moved towards the current lowest point $\mathbf{x}_1$. This step is called *multiple contractions*. This is computed from

$$\mathbf{v}_i = \mathbf{x}_1 - \sigma(\mathbf{x}_i - \mathbf{x}_1), \qquad i = 2, \cdots, n+1 \tag{8.8}$$

where σ is the shrinkage coefficient. This results in the new set of vertices $\mathbf{x}_1$, $\mathbf{v}_2$, $\cdots$,$\mathbf{v}_{n+1}$. This process is called *shrinkage* or *multiple contractions* and is illustrated in Fig. 8.4(d).

In our proposed method, equation (8.2) is adopted to create the initial simplex. Let n be the dimension of the design space. We start by randomly generating X_{random} points in the design space. The cost function is evaluated at each of the X_{random} points. The best point is retained and is assumed to be in the vicinity of the global optimum of the cost function. This process is shown in Fig. 8.5.

The initial simplex is created around the best point as follows. Assume that the best point, given by n dimensions, is $\mathbf{x}_{best}$ = (x_0, x_1, x_2, ... , x_{n-1}). $n+1$ points are generated by adding two units of length $\lambda = 2$ for each dimension, generating the following $N+1$ points:

$$\begin{aligned}
\mathbf{x}_0 = \mathbf{x}_{best} &= (x_0, x_1, x_2, ..., x_{n-1}) && (8.9)\\
\mathbf{x}_1 &= (x_0 + 2 \cdot e_0, x_1, x_2, ..., x_{n-1}) && (8.10)\\
\mathbf{x}_2 &= (x_0, x_1 + 2 \cdot e_1, x_2, ..., x_{n-1}) && (8.11)\\
\mathbf{x}_3 &= (x_0, x_1, x_2 + 2 \cdot e_2, ..., x_{n-1}) && (8.12)\\
\cdots &= \cdots\\
\mathbf{x}_n &= (x_0, x_1, x_2, ..., x_{n-1} + 2 \cdot e_{n-1}) && (8.13)
\end{aligned}$$

Then the gravity point is computed by averaging all the $n+1$ points:

$$\mathbf{x}_{gravity} = \frac{1}{n+1} \sum_{i=0}^{n} \mathbf{x}_i \tag{8.14}$$

The translation vector between the gravity point $X_{gravity}$ and the best point is computed

$$\Delta\mathbf{x}_{gravity} = \mathbf{x}_{gravity} - \mathbf{x}_{best} \tag{8.15}$$

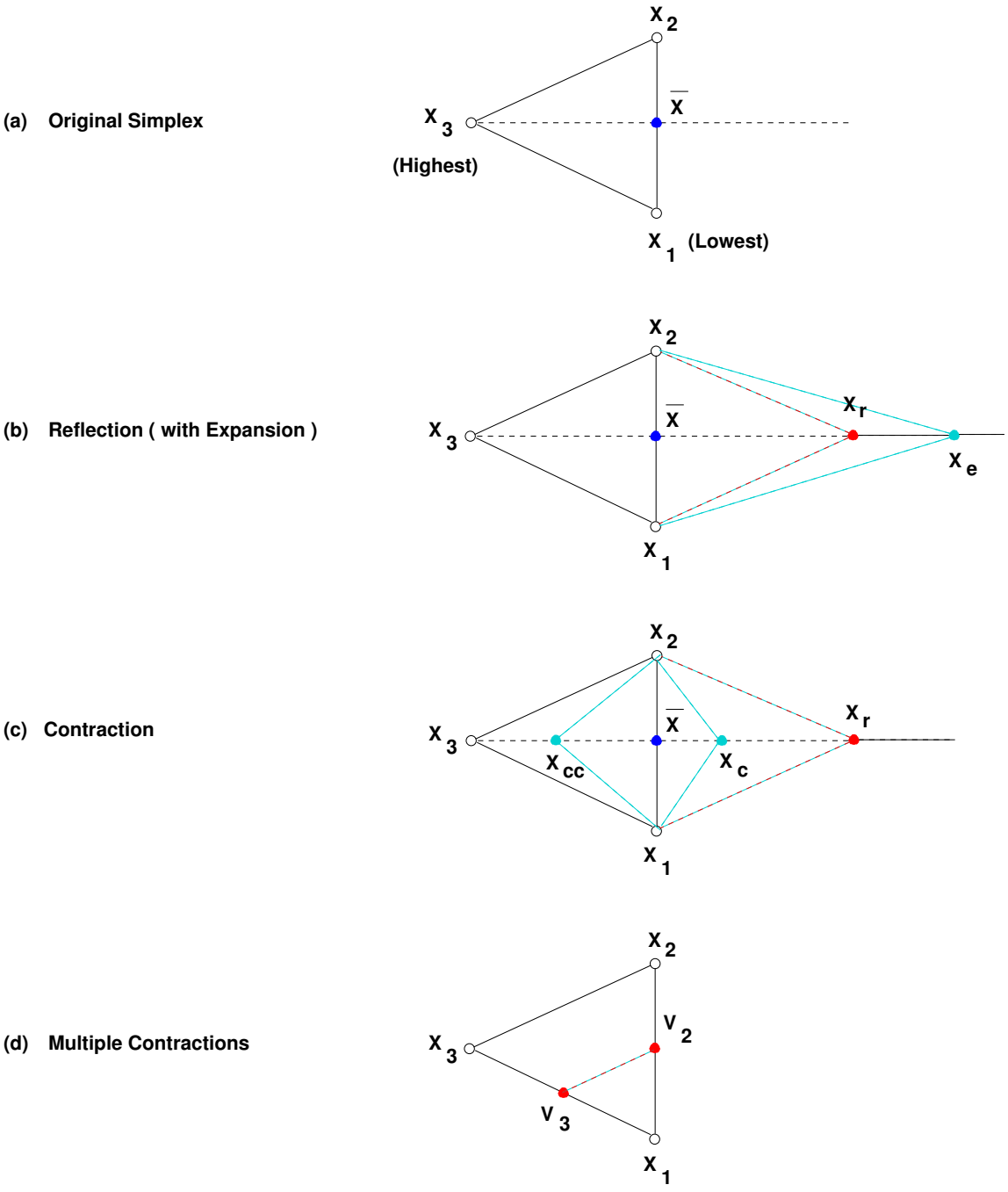

Figure 8.4: *Nelder-Mead Simplex Method*.

The $n+1$ points are moved so that the best point $\mathbf{x}_{best}$ is at their center of gravity using

$$\tilde{\mathbf{x}}_0 = \mathbf{x}_0 - \Delta\mathbf{x}_{gravity} \quad (8.16)$$

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 - \Delta\mathbf{x}_{gravity} \quad (8.17)$$

$$\tilde{\mathbf{x}}_2 = \mathbf{x}_2 - \Delta\mathbf{x}_{gravity} \quad (8.18)$$

$$\cdots = \cdots$$

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n - \Delta\mathbf{x}_{gravity} \quad (8.19)$$

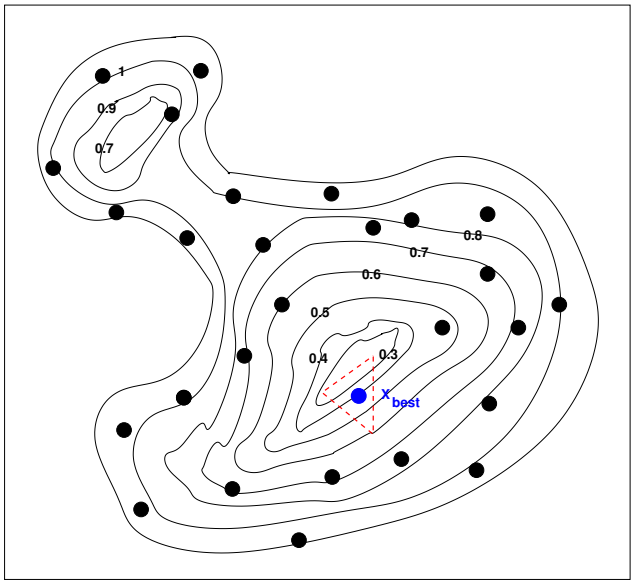

Figure 8.5: *Selection of the best point in the design space.*

An example is shown in Fig.8.6 for N=2.

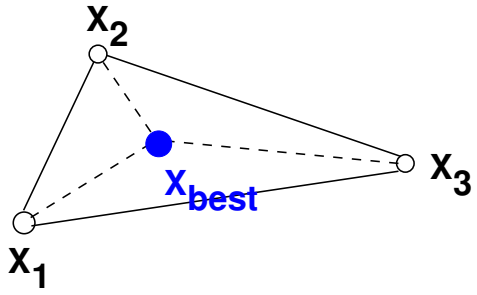

Figure 8.6: *For $n = 2$, a simplex consisting of the 3 points ($\mathbf{x}_1$, $\mathbf{x}_2$,$\mathbf{x}_3$) is created having $\mathbf{x}_{best}$ at its center of gravity..*

Once created, the initial simplex is evolved using the simplex method until it converges. At the end of convergence, the simplex size is sufficiently small and the solution point is chosen to be the average of $n + 1$ simplex points. Note that the above optimization method works well if the initial sampling of the design space is sufficiently dense, i.e. X_{random} is sufficiently large.

8.2.4 Definition of the Cost Function

The Nelder-Mead optimization function requires a cost function to evaluate at each of the $N + 1$ vertices of the simplex. The difficulty lies in how to express cost functions that incorporates both constraints and objectives. In general, the form of the cost function is difficult to determine since in many problems the solution space is very difficult to determine. It was proved in [Richardson89]

that a good cost function should incorporate the amount of violation, not only the number of evaluations. Following this principle, penalty functions [Richardson89, Coello00, Iskander03] are used to convert a constraint optimization problem (with constraints and objectives) into an unconstrained formulation. Suppose that the constraint optimization problem has the form:

$$\begin{aligned}
\min_i \quad & f_i(\mathbf{x}) < f_{target,i} && (8.20)\\
\max_j \quad & f_j(\mathbf{x}) > f_{target,j} && (8.21)\\
where \quad & g_k(\mathbf{x}) > A_k && (8.22)\\
& l_p(\mathbf{x}) < B_p && (8.23)\\
& h_m(\mathbf{x}) = D_m && (8.24)\\
& y_n(\mathbf{x}) \in [y_{n,min}, y_{n,max}] && (8.25)
\end{aligned}$$

where $\mathbf{x}$ is the parameter vector, f_i are the objective functions to minimize, $f_{target,i}$ are its minimum target values, f_j are the objective functions to maximize, $f_{target,i}$ are its maximum target values, g_k are the *greater-than* constraints, A_k are its lower bounds, l_p are the *less-than* constraints, B_p are its upper bounds, h_m are the *equality* constraints, D_m are its boundary values, y_n are the *range* constraints and $[y_{n,min}, y_{n,max}]$ are its boundary values. This problem can be solved using an unconstrained formulation having the form:

$$\begin{aligned}
\min_{i,j,k,p,m,n} \quad & \sum_i U_{min}(f_i(\mathbf{x})) && (8.26)\\
& + \sum_j U_{max}(f_j(\mathbf{x})) && (8.27)\\
& + \sum_k U_g(g_k(\mathbf{x})) && (8.28)\\
& + \sum_p U_l(l_p(\mathbf{x})) && (8.29)\\
& + \sum_m U_h(h_m(\mathbf{x})) && (8.30)\\
& + \sum_n U_y(y_n(\mathbf{x})) && (8.31)\\
& + \left(\sum_k N_g(g_k(\mathbf{x})) + \sum_p N_l(l_p(\mathbf{x})) + \sum_m N_h(h_m(\mathbf{x})) + \sum_n N_y(y_n(\mathbf{x})) \right)^2 && (8.32)\\
where \quad & U_{min},\ U_{max},\ U_g,\ U_l,\ U_h \text{ and } U_y \text{ are acceptability functions}\\
and \quad & N_g,\ N_l,\ N_h \text{ and } N_y \text{ are violation indicators}
\end{aligned}$$

In the following, we define the acceptability functions $U(.)$ and the violation indicators $N(.)$.

To do this, we use the bracket operator $< p, r >$ defined as:

$$< p, r >= \begin{cases} p^r & p > 0 \\ 0 & otherwise \end{cases} \tag{8.33}$$

This operator will be used to define the feasible regions where objectives and constraints have acceptable values. Now let us study the form of the acceptability functions adapted for every constraint type:

1. ***Greater-than***: In this type, the constraint value $g(\mathbf{x})$ is required to be greater than a lower bound value A, i.e. $g(\mathbf{x}) > A$. We define the acceptability function for this type to be:

$$U_g(g(\mathbf{x})) =< \frac{A - g(\mathbf{x})}{A}, 2 > \tag{8.34}$$

   This function returns a zero value only if $g(\mathbf{x}) > A$, otherwise it returns the square of the amount of the normalized violation (normalized by dividing it with the commensurate value A). The acceptability function is illustrated in Fig.8.7(a). In the figure, the acceptability function is shown for $r = 1$ and $r = 2$. The value of $r = 2$ is selected for this constraint type to reflect its importance w.r.t objective types which have $r = 1$, i.e. constraints should be satisfied before minimizing or maximizing objective functions.

2. ***Less-than***: In this type, the constraint value $l(\mathbf{x})$ is required to be less than an upper bound value B , i.e. $l(\mathbf{x}) < B$. We define the acceptability function for this type to be:

$$U_l(l(\mathbf{x})) =< \frac{l(\mathbf{x}) - B}{B}, 2 > \tag{8.35}$$

   This function returns a zero value only if $l(\mathbf{x}) < B$, otherwise it returns the square of the amount of the normalized violation. The acceptability function is illustrated in Fig.8.7(b).

3. ***Equality***: In this type, the constraint value $h(\mathbf{x})$ is required to be equal to a boundary value D, i.e. $h(\mathbf{x}) = D$. This constraint is equivalent to adding the two constraints:

$$h(\mathbf{x}) > (1 - \epsilon) \cdot D \tag{8.36}$$

$$h(\mathbf{x}) < (1 + \epsilon) \cdot D \tag{8.37}$$

   where ϵ tends to an infinitely small positive value. In this case, the acceptability function is defined to be the contribution of each of the constraints above. It has the following form:

$$U_h(h(\mathbf{x})) =< \frac{(1 - \epsilon) \cdot D - h(\mathbf{x})}{D}, 2 > + < \frac{h(\mathbf{x}) - (1 + \epsilon) \cdot D}{D}, 2 > \tag{8.38}$$

   This function returns a zero value only if $(1 - \epsilon) \cdot D < h(\mathbf{x}) < (1 + \epsilon) \cdot D$, otherwise it returns the squares of the amount of the normalized violation for each added constraint. The acceptability function is illustrated in Fig.8.7(c).

4. ***Range***: In this type, the constraint value $y(\mathbf{x})$ is required to be in the target range $[y_{min}, y_{max}]$, i.e. $y_{min} < y(\mathbf{x}) < y_{max}$. This constraint is equivalent to adding the two constraints:

$$y(\mathbf{x}) > y_{min} \tag{8.39}$$

$$y(\mathbf{x}) < y_{max} \tag{8.40}$$

In this case, the acceptability function is defined to be the contribution of each of the constraints above. It has the following form:

$$U_y(y(\mathbf{x})) =< \frac{y_{min} - y(\mathbf{x})}{y_{min}}, 2 > + < \frac{y(\mathbf{x}) - y_{max}}{y_{max}}, 2 > \tag{8.41}$$

This function returns a zero value only if $y_{min} < y(\mathbf{x}) < y_{max}$, otherwise it returns the squares of the amount of the normalized violation for each added constraint. The acceptability function is illustrated in Fig.8.7(d).

Objectives functions are expressed in the cost function using the same formulation defined for constraints. Two types of objectives exist:

1. ***Maximize***: In this type, the objective value $f(\mathbf{x})$ is required to be greater than a target value f_{target}, i.e. $f(\mathbf{x}) > f_{target}$. We define the acceptability function for this type to be:

$$U_{max}(f(\mathbf{x})) =< \frac{f_{target} - f(\mathbf{x})}{f_{target}}, 1 > \tag{8.42}$$

This function returns a zero value only if $f(\mathbf{x}) > f_{target}$, otherwise it returns the amount of the normalized violation (normalized by dividing it with the commensurate value f_{target}). The acceptability function is illustrated in Fig.8.7(a) for $r = 1$. The value of $r = 1$ is selected for this objective type to reflect its lower importance w.r.t constraints which have $r = 2$, i.e. constraints should be satisfied before minimizing or maximizing objective functions.

2. ***Minimize***: In this type, the objective value $f(\mathbf{x})$ is required to be less than a target value f_{target} , i.e. $f(\mathbf{x}) < f_{target}$. We define the acceptability function for this type to be:

$$U_{min}(f(\mathbf{x})) =< \frac{f(\mathbf{x}) - f_{target}}{f_{target}}, 1 > \tag{8.43}$$

This function returns a zero value only if $f(\mathbf{x}) < f_{target}$, otherwise it returns the amount of the normalized violation. The acceptability function is illustrated in Fig.8.7(b) for $r = 1$.

The above discussion computes the amount of violation for each constraint and objective type. To compute the violation count for constraint types, we define the function $N(.)$ to be

$$N(C(\mathbf{x})) = \begin{cases} 1 & C(\mathbf{x}) \; is \; not \; satisfied \\ 0 & otherwise \end{cases} \tag{8.44}$$

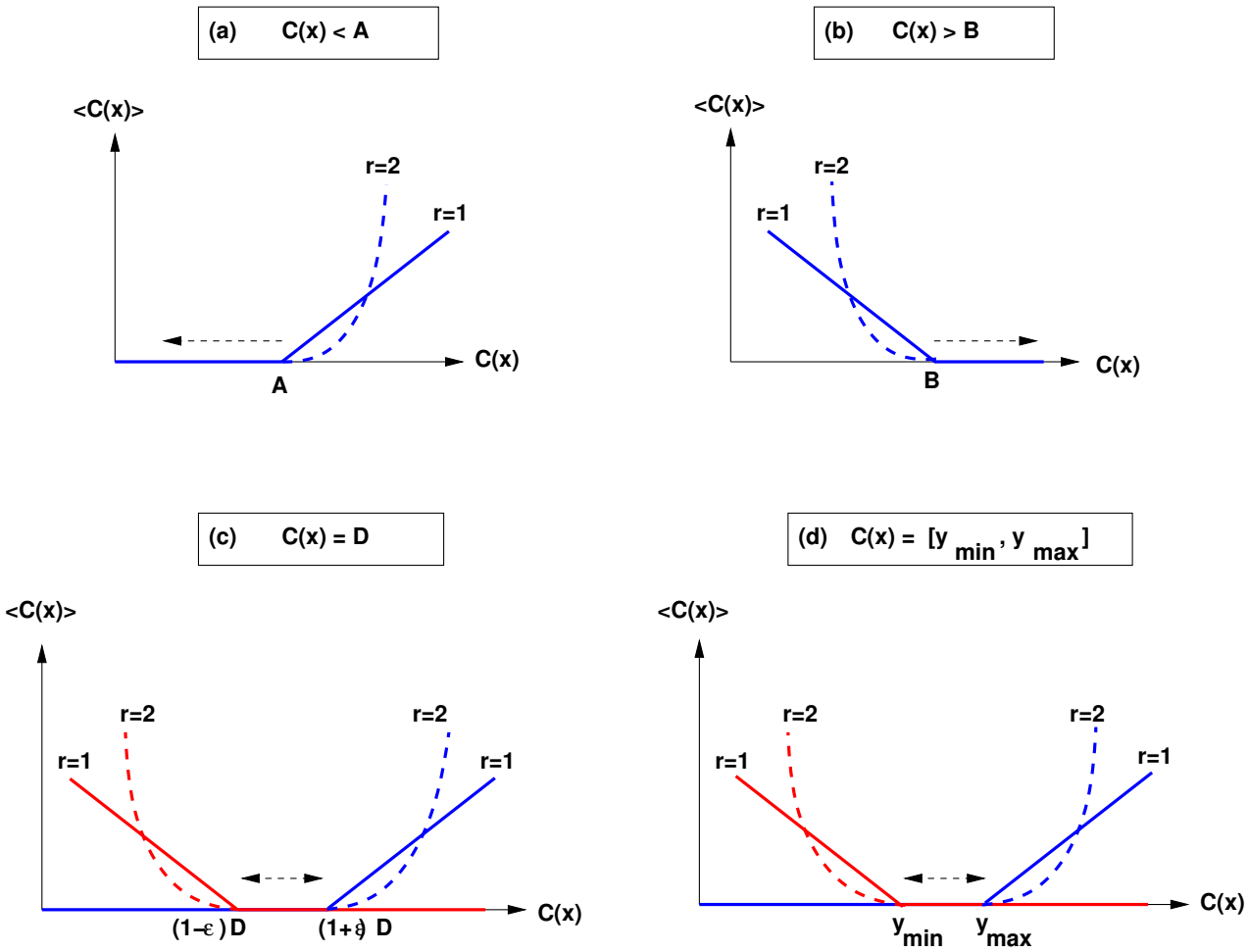

Figure 8.7: *Acceptability functions: (a) Greater-than type, (b) Less-than type, (c) Equality type, and (d) Range type. Dotted arrows point to feasible regions of interest.*

where $C(\mathbf{x})$ represent the constraint function. The square of the violation count is added for all the constraints (without objectives) in order to make a pressure on the optimizer to compute first feasible regions, i.e. where all the constraints are satisfied without objectives. This way, objectives are considered of less importance than constraints by using lower exponent for objective violation ($r = 1$) and by adding the square of violation count only for constraints.

8.2.5 Optimizing an Analog IP

In order to optimize an analog IP, a testbench has to be configured. The testbench, shown in Fig. 8.8 consists of five main parts:

1. **Analog IP instantiation**: The analog IP is instantiated inside the testbench. The analog IP

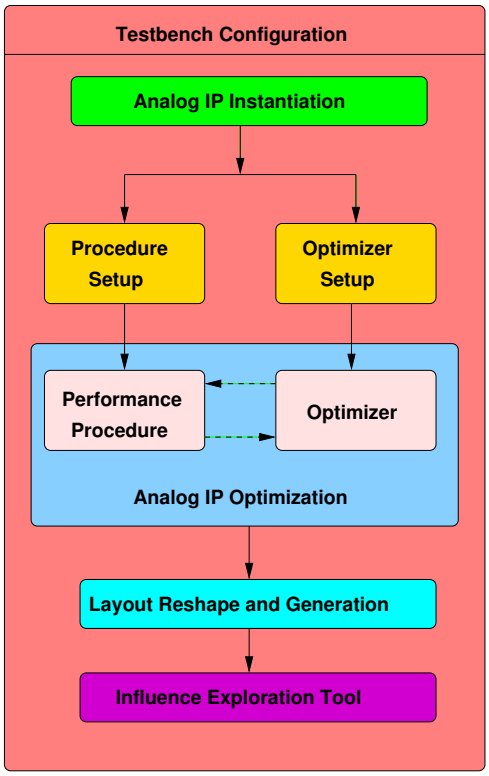

Figure 8.8: *Testing an analog IP*.

is accessible through its input/output parameters and its procedures, especially the performance procedure coded by the designer to evaluate circuit performances.

2. **Optimizer and performance procedure setup**: The optimization variables, representing some analog IP parameters, are selected . Their ranges of variation are specified. Constraints and objectives are added to the optimizer setup. The optimizer is then created at the root level of the analog IP and bound to procedure computing performances called *performance procedure*.

3. **Analog IP optimization**: The analog IP is optimized by executing the optimizer in a loop with the performance procedure. This results in altering some input parameters in the analog IP in order to satisfy objectives and constraints. At the end, a solution is achieved and the resulting analog IP instance is preserved.

4. **Layout reshape and generation**: The layout is generated for the preserved analog IP.

5. **Influence exploration tool**: The influence exploration tool is displayed for the designer to allow the exploration and characterization of the preserved analog IP.

In the following section, the *Application Program Interface API* used to perform optimization setup and execution will be explained in further details.

8.2.6 API for Knowledge-Aware Synthesis

In order to configure an analog IP for the purpose of optimization, an *Application Program Interface API* has been developed. Table. 8.1 gives the C/C++ macros definitions consisting the API. The source code for knowledge-aware synthesis for the OTA amplifier is listed in appendix H. This code is used to produce the results in section 8.3.

8.3 Results

8.3.1 Synthesizing an Analog IP

In this subsection, our proposed methodology for optimization, shown in Fig. 8.3, is used to synthesize the two-stage amplifier shown in Fig. 8.9.

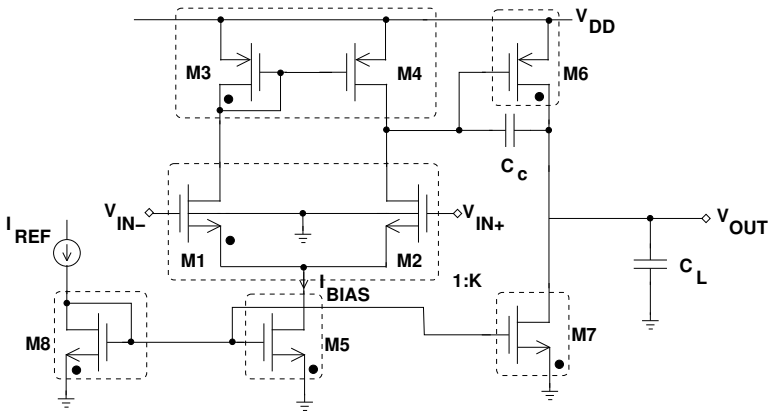

Figure 8.9: *Two-stage amplifier*.

After applying it to the amplifier, the module dependency graph of the amplifier is obtained as shown in Fig. 8.10. This graph assumes :

1. The amplifier is to be designed for nonzero systematic offset, i.e $V_{D,M_3} \neq V_{G,M_6}$.
2. $W_{M8} = W_{M5}$.
3. $L_{M5} = L_{M7} = L_{M8} = L_{\{M8,M5,M7\}}$.

To perform optimization, some parameters have been fixed such as $TEMP$, V_{DD}, V_{SS}, V_{ICM}, V_{OUTCM}. Other parameters have been allowed to vary by the optimizer such as I_{BIAS}, $V_{eg,CM}$, L_{CM}, L_{M_6}, V_{eg,M_6}, $V_{eg,DP}$, L_{DP}, V_{eg,M_5}, $L_{\{M_8,M_5,M_7\}}$, K. In addition, the compensation capacitance C_C was allowed to vary.

We recall the amplifier graph shown in Fig. 6.34 for convenience. In the following, we analyze the graph and explain how it sizes and biases the two-stage amplifier:

1. The amplifier parameters needed to execute the graph are: $TEMP$, V_{DD}, V_{SS}, I_{BIAS}, $V_{eg,CM}$, L_{CM}, L_M6, V_{eg,M_6}, $V_{eg,DP}$, L_{DP}, V_{ICM}, V_{OUTCM}, $V_{eg,M5}$, $L_{\{M8,M5,M7\}}$ and K. These are represented by *rectangle* nodes.

2. The variables and parameters used for parameter mapping are represented as *thin circle* nodes. As an example, variable (C2,veg_{cm},61) maps the amplifier parameter (C1,$V_{eg,CM}$,67) into the current mirror parameter (C3,$V_{eg,CM}$,8).

3. Device parameters are propagated to transistors forming the device. As an example, the current mirror parameter (C3,$V_{eg,CM}$,8) is propagated to M_3 via (C4,$V_{eg,M3}$,7) and to M_4 via (C4,$V_{eg,M4}$,7). Note that M_3 and M_4 share the same effective gate-source voltage (C4,V_{eg},7).

4. Transistor widths are computed in nodes (C8,W_{M_1,M_2},21), (C8,W_{M_3,M_4},11), (C8,W_{M_6},31), (C7,W_{M_5,M_8},36) and (C8,W_{M_7},25).

5. Since $W_{M8} = W_{M5}$, they share the same node (C7,W_{M_5,M_8},36).

6. Since $V_{D,M_3} \neq V_{G,M_6}$ is imposed for nonzero systematic offset, they are split to nodes (C5,{V_{D,M_1},2) and (C7,V_{G,M_6}},1).

7. The design plan is represented by the graph as follows:

   (a) $V_{G/D,M3}$ is computed using *OPVGD($V_{eg,M3}$)* in node (C5,$V_{G/D,M3}$,2).

   (b) $V_{D,M3}$ is used to compute $V_{D,M5}$, which is the same as $V_{S,M1}$, using *OPVS($V_{eg,M1}$,$V_{B,M1}$)* in node (C6,$V_{S,M1}$,24).

   (c) $V_{D,M5}$ is then used to compute $V_{G,M5}$ using *OPVG($V_{eg,M5}$)* in node (C7,$V_{G,M5}$,29).

   (d) $V_{G,M5}$ is used to compute I_{DS,M_8} using $OPIDS(V_{G,M_8}, V_{S,M_8})$ in node (C8,I_{DS,M_8},41).

The optimization loop of Fig. 8.3 is executed using the amplifier module dependency graph of Fig. 8.10. The optimization is performed with the number of parameters $N = 9$ shown in Table 8.2, the initial points $X_{random} = 100$ and a constant $\lambda = 2$. Performance equations have been extracted from OCEANE [Porte08] and manually coded inside the performance procedure. Table 8.2 shows the target specifications of the amplifier, the synthesis and the simulation results in 130nm CMOS technology. The average runtime on an Intel Centrino 1.7GHz with 2MB Cache is 76 seconds. This runtime is due to the small ranges of variation of the optimization variables. These small ranges relief us from the burden of using sophisticated optimization algorithms. Moreover, the simulation results agree with the synthesized design. The results show that our approach produces simulator-like accurate designs with a very reasonable amount of time, thus allowing interactive design of analog circuits. It is important to mention that the synthesis is performed

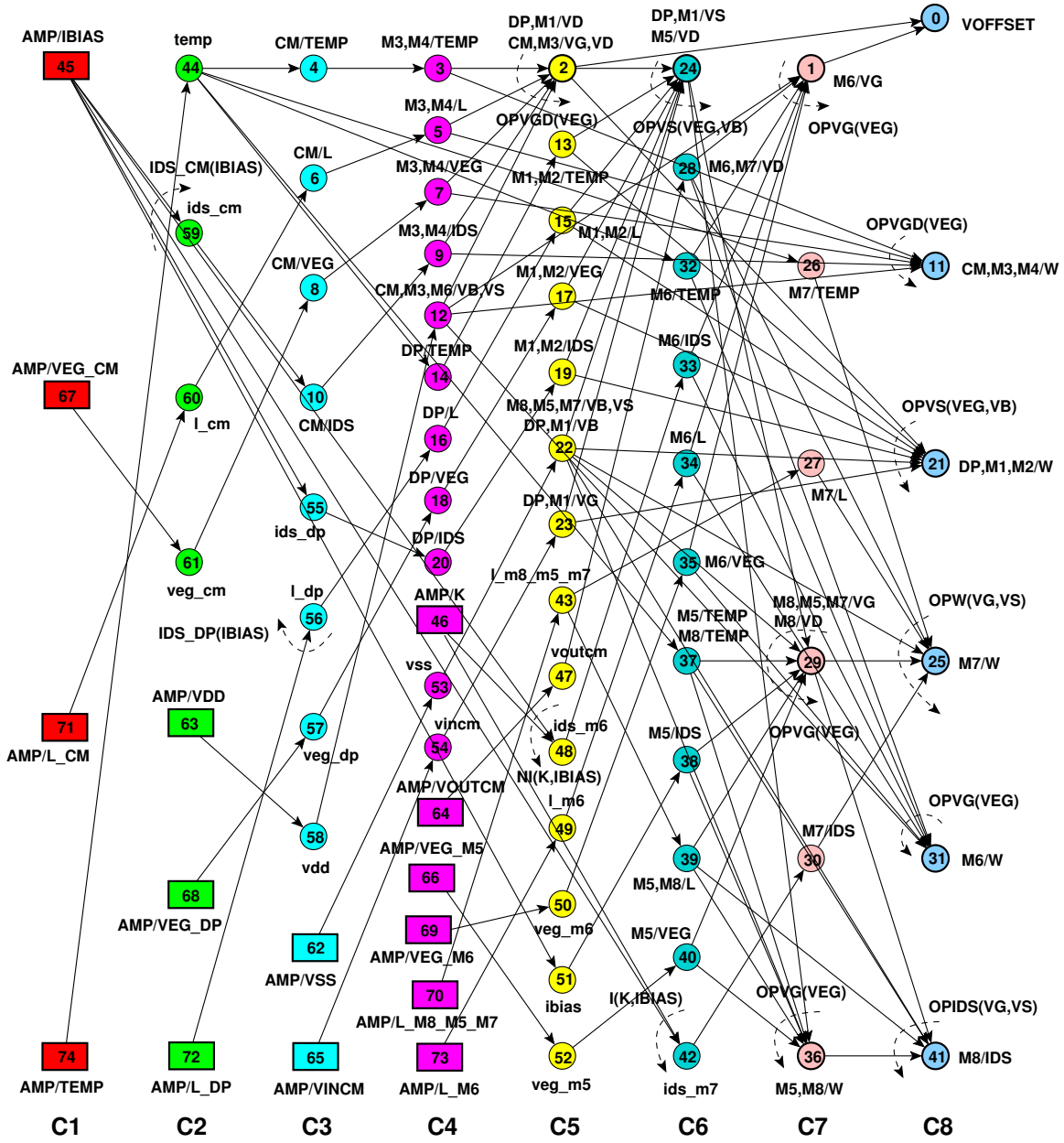

Figure 8.10: *Module dependency graph of the amplifier for nonzero systematic offset in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

in the designer mode and is coupled with a layout generation phase which truncates of widths to the physical grid. Therefore, some differences exist between the simulation and synthesis. In addition, some inaccuracy exists in performance equations.

8.3.2 Comparison to the State-of-Art

The measured runtime for the amplifier is compared against state-of-art knowledge-based and simulation-based analog synthesis systems published during the last two decades. The comparison is summarized in table 8.3. The obtained runtime is found to be better over previously published work, except for OASYS [Harjani87, Harjani88, Harjani89b, Harjani89a]. OASYS is considered the fastest since knowledge is manually coded and optimized inside the synthesis tool. AMGIE [Plas01], which adopts similar strategies is in the same order of speed. The highest runtime is obtained in the case of simulation-based synthesis systems like ASTRX/OBLX [Ochotta96], ASF [Krasnicki01] and ANACONDA [Phelps00]. Our proposed methodology proves to be efficient compared to other state-of-art synthesis systems.

8.3.3 Changing the Specifications

Since an analog IP is a reusable building block, it is easy to re-synthesize it using different specifications. Suppose that the unity gain frequency is to be increased by at least a factor of 2. For this purpose, the range of I_{BIAS} is doubled. Table 8.4 shows the synthesis versus simulation results for this case. The unity gain frequency is required to be above 12Mhz for a doubled I_{BIAS} that can reach 60μA. The results show that I_{BIAS}, hence I_{BIAS} has doubled and the unity gain frequency has increased compared to table 8.2. The average runtime on Intel Centrino 1.7GHz with 2MB is 87 secs. This proves that the proposed methodology can be used for interactive design efficiently.

8.4 Conclusions

The hierarchical sizing and biasing have been introduced into an optimization loop. It automatically generates design plans which reduce the design space of the circuit by using optimization variables that have narrow ranges of variation. The results show that our approach produces simulator-like accurate designs with a very reasonable amount of time, thus allowing interactive design of analog circuits. Our approach is in good position compared to the state-of-art knowledge-based and simulation-based synthesis tools previously published. The proposed methodology have been called *Knowledge-aware synthesis* since knowledge is automatically extracted for the analog circuits and applied directly into the optimization loop.

Macro	Definition
CAIRO_SET_PARAM(”<inst-name>”, ”<param-name>”, <value>)	This macro is called to set an input parameter, where <inst-name> is the analog IP instance name, <param-name> is the parameter name and <value> is the floating-point value given to the parameter.
CAIRO_SET_PARAM_DOMAIN(”<inst-name>”, ”<param-name>”, <min-value>, <max-value>, <step>)	This macro is called to set the range of variation for an input parameter where <inst-name> is the analog IP instance name, <param-name> is the parameter name, <min-value> is the minimum permissible value, <max-value> is the maximum permissible value and <step> is the step of change.
CAIRO_SET_PARAM_CONSTRAINT(”<inst-name>”, ”<param-name>”, [CP_ABOVE | CP_BELOW | CP_EQUAL | CP_WITHIN],<value> [, <value>])	This macro is called to set a constraint for a performance parameter, where <inst-name> is the analog IP instance name, <param-name> is the parameter name, [CP_ABOVE | CP_BELOW | CP_EQUAL | CP_WITHIN] is a constraint type, and <value> is the boundary value. For the constraint types: CP_ABOVE or CP_BELOW or CP_EQUAL, one boundary value is specified. For the constraint type CP_WITHIN, two boundary values specify the range of variation.
CAIRO_OPTIMIZER_USE(”<inst-name>”)	This macro is called to create an optimizer for the analog IP, where <inst-name> is the analog IP instance name.
CAIRO_OPTIMIZE_PROCEDURE(”<inst-name>”, ”<proc-name>”)	This macro is called to set the name of the procedure used to compute performances, where <inst-name> is the analog IP instance name and <proc-name> is the name of the performance procedure. The procedure computes all the performances shown in the vector in Fig. 8.3.
CAIRO_OPTIMIZE(”<inst-name>”,<pop-length>, [CP_DESIGN | CP_SYNTHESIS])	This macro is called to start the optimization process by specifying the length of population and the synthesis mode, where <inst-name> is the analog IP instance name, <pop-length> is the length of the population of random points and [CP_DESIGN | CP_SYNTHESIS] is the execution mode. If CP_DESIGN is specified, the analog IP dependency graph is not evaluated and the performance procedure is called for performance evaluation. Otherwise, if CP_SYNTHESIS is specified, then the analog IP dependency graph is evaluated before calling the performance procedure.
CAIRO_OPTIMIZER_RELEASE(”<inst-name>”)	This macro is called to free the optimizer, where <inst-name> is the analog IP instance name.

Table 8.1: *Macros definitions for knowledge-aware synthesis*.

Table 8.2: *Synthesis Results in 130nm CMOS with V_{DD} = 1.2V in Designer Mode.*

Circuit Performances	Target	Synthesis (with Layout)	Simulation
Dynamic Gain (dB)	> 65	71.22	70.83
Common-Mode Gain (dB)	< 17	5.59	8.82
Unity Gain Frequency (MHz)	> 6	10.79	10.66
Phase Margin (degrees)	> 76	76.69	76.9
Input Offset Voltage (mV)	< 2	-1.03	-1.07
Input Noise @1Hz ($\mu V/\sqrt{Hz}$)	< 20	10.86	10.14
Input Noise @UGF ($\mu V/\sqrt{Hz}$)	< 0.02	0.014	0.015
Slew Rate ($V/\mu s$)	> 6	8.25	9.16
Transistors in Saturation	$= 8$	8	8
Power (mW)	< 1	0.176	0.176
Common-Mode Rejection Ratio (dB)	–	65.63	62.01
Minimum Common-Mode Input (V)	–	0.53	0.53
Maximum Common-Mode Input (V)	–	1.04	1.04
Maximum Output Voltage (V)	–	1.12	1.12
Minimum Output Voltage (V)	–	0.10	0.10
Area (μm^2)	–	69	69
Average Runtime over 10 runs (secs)	-	76	-
Variable Parameter	Domain	Synthesis	
$L_{M_5} = L_{M_7} = L_{M_8}(\mu m)$	0.2:3:0.1	0.6	
$L_{CM}(\mu m)$	0.2:3:0.1	0.6	
$L_{M6}(\mu m)$	0.2:3:0.1	0.2	
$L_{DP}(\mu m)$	0.2:3:0.1	0.8	
V_{eg,M_5} (V)	0.01:0.2:0.01	0.1	
$V_{eg,DP}$ (V)	0.01:0.2:0.01	0.08	
$V_{eg,CM}$ (V)	-0.2:-0.01:0.01	-0.15	
V_{eg,M_6} (V)	-0.2:-0.01:0.01	-0.04	
$I_{BIAS}(\mu A)$	10:30:1	25.0	
K	1:5:0.5	4.5	
C_C (pF)	1:5:0.1	2.9	
Constant Parameter	Value	Synthesis	
$TEMP$ (degrees)	300.15	300.15	
V_{DD} (V)	1.2	1.2	
V_{SS} (V)	0.0	0.0	
V_{ICM} (V)	0.6	0.6	
V_{OUTCM} (V)	0.6	0.6	
C_{LOAD} (pF)	3.0	3.0	

Table 8.3: *Comparison With State of Art Synthesis Tools*.

Tool	Performance Evaluator	Synthesis Time	Machine
OASYS [Harjani89b]	Equations	5 seconds	VAX 8800
OCEANE[Porte08]	Equations	1 minute	Centrino 1.7GHz, 2MB Cache
This Work	Equations	76 seconds	Centrino 1.7GHz, 2MB Cache
OPASYN [Koh90][1]	Equations	1 minute	VAX 8800
AMGIE [Plas01]	Equations	Few minutes	SUN Ultra 1-170
ASTRX/OBLX [Ochotta96]	Asymptotic Waveform simulator	16 minutes	IBM RS/6000-550
ASF [Krasnicki01]	Spice Simulator	74 minutes	10 Ultra Sparc Solaris
ANACONDA [Phelps00]	TI Spice Simulator	2.8 hours	16 (300 Mhz) Ultra 10 + 4 (300 Mhz) dual-processor Ultra 2

1. OPASYN requires simulation for fitting performances.

Table 8.4: *Synthesis Results in 130nm CMOS with V_{DD} = 1.2V in Designer Mode.*

Circuit Performances	Target	Synthesis (with Layout)	Simulation
Dynamic Gain (dB)	> 65	72.04	71.49
Common-Mode Gain (dB)	< 17	15.91	18.13
Unity Gain Frequency (MHz)	$> \underline{12}$	17.2	17.09
Phase Margin (degrees)	> 76	76.44	76.59
Input Offset Voltage (mV)	< 2	-0.67	-0.6882
Input Noise @1Hz ($\mu V/\sqrt{Hz}$)	< 20	8.31	7.02
Input Noise @UGF ($\mu V/\sqrt{Hz}$)	< 0.02	0.0125	0.0127
Slew Rate ($V/\mu s$)	> 6	18.61	20.0
Transistors in Saturation	$= 8$	8	8
Power (mW)	< 1	0.432	0.432
Common-Mode Rejection Ratio (dB)	–	56.13	53.36
Minimum Common-Mode Input (V)	–	0.58	0.58
Maximum Common-Mode Input (V)	–	1.03	1.03
Maximum Output Voltage (V)	–	1.1	1.1
Minimum Output Voltage (V)	–	0.09	0.09
Area (μm^2)	–	155	155
Average Runtime over 10 runs (secs)	-	87	-
Variable Parameter	Domain	Synthesis	
$L_{M_5} = L_{M_7} = L_{M_8}(\mu m)$	0.2:3:0.1	0.3	
$L_{CM}(\mu m)$	0.2:3:0.1	0.9	
$L_{M6}(\mu m)$	0.2:3:0.1	0.3	
$L_{DP}(\mu m)$	0.2:3:0.1	1.8	
V_{eg,M_5} (V)	0.01:0.2:0.01	0.07	
$V_{eg,DP}$ (V)	0.01:0.2:0.01	0.17	
$V_{eg,CM}$ (V)	-0.2:-0.01:0.01	-0.17	
V_{eg,M_6} (V)	-0.2:-0.01:0.01	-0.09	
$I_{BIAS}(\mu A)$	10:$\underline{60}$:1	56	
K	1:5:0.5	5	
C_C (pF)	1:5:0.1	2.6	
Constant Parameter	Value	Synthesis	
$TEMP$ (degrees)	300.15	300.15	
V_{DD} (V)	1.2	1.2	
V_{SS} (V)	0.0	0.0	
V_{ICM} (V)	0.6	0.6	
V_{OUTCM} (V)	0.6	0.6	
C_{LOAD} (pF)	3.0	3.0	

Chapter 9

Conclusion and Future Directions

9.1 Conclusion

The complexity of integrated electronic circuits being designed is continuously increasing as advances in process technology make it possible to create mixed-signal integrated SoC designs. To manage such huge complexity, design reuse tools and methodologies need to be developed to deal with high levels of complexity. Since AMS blocks cannot be easily synthesized from a high-level specification without low-level support, AMS-IP blocks still represent a real bottleneck in the mixed-signal design process of SoC designs. It is believed that firm intellectual properties (IP) are the most appropriate format to represent AMS-IP blocks. It is clear that the identification of an intermediate design representation for firm IPs acquires analog design automation tools lots of insight to deal with different problems of analog circuit design.

In this context the methodology presented in this thesis work focuses on departing from a firm IP unsized schematic and automatically generate and store its design knowledge in an appropriate design representation. The propose methodology is fivefold:

Transistor sizing and biasing methodology: In this part, we developed a methodology for accurately sizing and biasing a MOS transistor based on the BSIM3V3 transistor model. The concept of the *sizing and biasing operators* have been introduced for this purpose. Each operator compute an unknown electrical parameter by numerically inverting the standard BSIM3V3 MOS transistor model. The sizing and biasing operators have been implemented in CAIRO+ framework. These have been introduced to enhance the architecture of the MOS engine. The methodology proved its precision and efficiency in sizing and biasing MOS transistors.

Device sizing and biasing methodology: In this part, we developed a methodology for defining basic building blocks called *devices*. This step is essential to tackle the problem of hierarchy in analog design. Some key concepts like *the transistor packing*, *the reference transistor*, *the device constraints* have been introduced. These concepts have been used to automatically generate design plans for

devices and represent design plans using *device dependency graphs*. The methodology proved to be efficient and accurate in sizing and biasing devices.

Circuit sizing and biasing methodology: In this part named *hierarchical sizing and biasing*, we developed a methodology for automatically generating suitable design plans for complex circuit topologies. Analog firm IP are first described as unsized schematic. Then the methodology automatically extracts design plans for the whole circuit topology by merging dependency graphs of devices and lower-level modules. The mering process is done in a bottom-up fashion. The resulting circuit design plan is stored in *module dependency graphs*. The circuit is then sized and biased by evaluating the module dependency graph in a top-down fashion. The methodology also dealt with different aspects in analog design, namely, *incomplete knowledge* in *under-specified designs*, *systematic offset* in *over-specified designs* and *negative feedback circuits*. The methodology have been successfully applied to different analog firm IP intellectual properties and proved it precision and efficiency in sizing and biasing complete analog firm IPs.

Knowledge-aware synthesis methodology: This part has been developed as an application to prove the efficiency of the proposed methodologies. This was demonstrated by introducing our *hierarchical sizing and biasing methodology* inside an optimization loop using the *Nelder-Mead Simplex* as it search method. It was shown that this step contributed to a considerable speed-up in optimization time, since the designs that have been generated by the optimizer tended to be feasible. In addition, the methodology allowed the use of more intuitive set of optimization variables consisting of voltages, lengths and currents. These variables have narrow ranges of variation. This led to a reduction in the design space of the optimized circuits. Consequently, the execution time obtained was very reasonable allowing interactive designs of analog circuits. Moreover, the generated designs had the same quality of a simulator-generated designs. Finally, the proposed synthesis platform was successfully used to migrate an analog IP for different specifications.

Case studies: In order to explore the effectiveness of the proposed methodologies, four case studies have been investigated: *a fully differential cascode current-mode integrator*, *a fully differential common-mode feedback amplifier*, *a fully differential transconductor* and *a fully differential body-input operational amplifier*. Each case study presented different class of analog design problems. The problem of *incomplete knowledge* has been successfully illustrated for the fully differential cascode current-mode integrator. The problem of *systematic offset* and *negative feedback* have been successfully illustrated for the fully differential common-mode feedback amplifier. The problem of very complex analog design hierarchies has been successfully demonstrated for the fully differential transconductor. Finally, the problem of circuit sizing in nanotechnology, very low-power design and uncommon circuit interconnections have been successfully demonstrated for the fully differential body-input operational amplifier. In all the case studied presented, the hierarchical sizing

and biasing methodology has successfully synthesized the analog firm IPs.

9.2 Future Work

The presented work has tackled different aspects in analog design automation. Different research directions have emerged from the type of problems explored. In particular, the following points seem interesting to investigate:

Transistor sizing and biasing: In order to adapt the proposed work to explore issues in nanotechnology, more advanced compact models like PSP, BSIM4, ... should be integrated. It is important also to deal with passive elements like resistors since they conduct current in steady state. Therefore, a resistor engine based on industrial level compact models should be developed.

Device sizing and biasing: The devices implemented so far are based on the MOS transistor. More complex devices that are based on passive elements, e.g. matched resistor arrays, should be integrated with their intrinsic constraints.

Circuit sizing and biasing: This part can be enhanced as follows:

- Instead of asking the designer to document Kirchhoff's current and voltage laws. These equations should be automatically extracted as in analog simulator. On the other hand, documenting these equations makes the designers think about how their circuits operate, rather than allowing them to simulate blindly. This should be an option not a requirement.
- The circuit sizing and biasing may produce non-physical results if the parameters supplied are not correct. Therefore, strong error detection strategies should be put in place so that a designer would like to feel that he/she is dealing with program that understands circuits.
- The methodology deals only with DC analysis, DC sweep and small-signal analysis. More complex analysis types such as transient analysis, Monte-Carlo analysis, mismatch analysis, sensitivity analysis, ... , should be implemented. The execution time for these analyses should be optimized.
- The methodology should integrate worst-case analysis and tolerance analysis techniques to ensure robustness of produced designs.
- The methodology should answer designer questions like: why is the exact value of a parameter he started with, and the resulting exact value of the same parameter, is useful ?

Knowledge-aware synthesis: This part can be enhanced as follows:

- More search algorithms need to be integrated such as genetic algorithms, simulated annealing, ... etc.
- Instead of using random starting points at the initial population, uniformly distributed population members can be generated.

Tool usage: The following points are considered as future enhancements that relates tool architecture and usage:

- The program lacks schematic capture, and in general a mature graphical interface. Designers are now used to such interfaces and they would expect them of any program they are likely to use. Therefore, a *Cadence* interface would be desirable.
- Diagnostic messages need to be easy-to-understand diagnostics. Engineers at companies are short of time, and any such delay is likely to be viewed as a serious drawback.
- During design documentation, an engineer often modifies the circuit, e.g. adding a transistor, removing a resistor, etc. This would create a new topology, and it would have to be debugged and compiled again. This should be compared against the ease of doing the above using *Spice* or *Cadence*. Consequently, the language issues should be highly reduced.
- The program should plot virtually anything versus anything else. For example, one could ask it to plot the voltage at a node versus the width of a transistor. Some viewing capabilities should be integrated to visually explore results.
- The tool usage need to be compared against some modern programs like *MATLAB* and the simulation-based synthesis tool *NeoCircuit*.

Future projects: Many research projects are now emerging as a direct application of the proposed methodology:

- **MOCSA project:** This project is a two year research project that started with a collaboration with *Pierre-Fouilhoux* and *Safia-Kedad Sidhom* of the LIP6/DESIR/RO team. The project aims at optimally modeling knowledge for analog systems. MOCSA stands for *Modélisation Optimale par la Connaissance des Systèmes Analogiques*. The project was financed in 2007 and is accepted for finance extension for 2008.
- **Technology migration:** A collaboration is established with *Nöelle Lewis* from IMS, Bordeaux to integrate the technology migration algorithm inside CAIRO+. This algorithm scales a circuit in a newer technology depending on a figure of merit defined by the designer. This can be used to eliminate the generation of initial population. Hence, optimize around the scaled solution and decrease considerably optimization time.

- **Standardization efforts** for SystemC-AMS language are now progressing. The proposed methodology can define a new model of computation that is adapted for SystemC-AMS language. This allows the documentation and execution of a system described fully in SystemC-AMS. Synthesis and simulation tasks are then conducted using the unified platform of SystemC-AMS.
- **Performance modeling:** This field is very complementary with the proposed methodology. Since performance equations should be supplied by the designer, several performance modeling techniques can be introduced to fully automate the design process of analog intellectual properties. These methods includes influence exploration, response surface modeling, support-vector machines, behavioral model generation, ... etc.
- **System level modeling:** A system level modeling should be developed to allow expressing design constraints and performance equations in the system level. This way, a complete design representation from system level to performances is made possible.

Appendix A

Second-Order Effects in Deep Submicron

Since our proposed method for transistor sizing and biasing relies on identifying design errors in early design stages, common sources of error have to be identified. Years ago, long-channel devices were used in circuit design. Since transistor lengths were big, the dependence of the threshold voltage V_{th} on the length was neglected and V_{th} was considered constant. Today this assumption is no longer valid for deep-submicron technologies. For shrinking devices, V_{th} is becoming more and more dependant on device geometries and biases. However, designers may still assume constant V_{th}, during hand calculations. To eliminate such sources of errors, the effects impacting the threshold voltage should be taken into account in automated design flows. In subsequent sections, the effects impacting the threshold voltage will be discussed [Cheng99].

A.1 Normal Short-Channel Effects

The threshold voltage of a long channel device (with channel length ($L \sim 10\mu m$) is independent of the channel length L and the drain voltage V_d. However, experimentally it is observed that V_{th} decreases as L decreases or V_d increases, as shown in Fig. A.1. This effect is called *the short channel effect* [Duvvury86]. In this figure, the normal short channel effect, that is, V_{th} decreases monotonically as the channel length L decreases. It s also known as the V_{th} roll-off. The dependence of V_{th} on L and V_{ds} in short-channel devices cannot be ignored. If the value of V_{th} drops greatly as the L and V_{ds} vary, the device may exhibit excessive drain leakage current even when V_{gs} = 0V [Chan87].

A.2 Reverse Short Channel Effects

In devices using halo or packet implantation, it has been found that, as shown in Fig. A.2, V_{th} initially increases with decreasing channel length. This is called *the reverse short channel effect (RSCE)*, or V_{th} roll-up [Nishida81]. V_{th} reaches a maximum value at a certain channel length, and as L decreases further, V_{th} starts to decrease . This last phenomenon is called the V_{th} roll-off. The com-

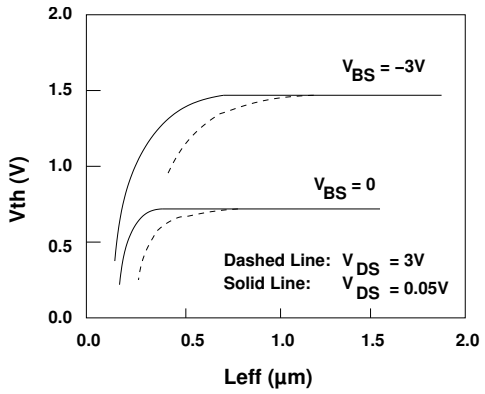

Figure A.1: *Profile of V_{th} versus L_{eff} for normal short-channel effects.*

bined *RSCE* and V_{th} roll-off effects result in a *hump* in the characteristics of V_{th} versus L, as shown in Fig.A.2. The cause of *RSCE* is the non-uniform lateral doping [Hanafi93]. For some technologies such as pocket implantation the channel doping concentration near the source an drain is higher than in the middle of the channel. The increased doping concentration in these regions can result in an increase in V_{th} if the channel length becomes small.

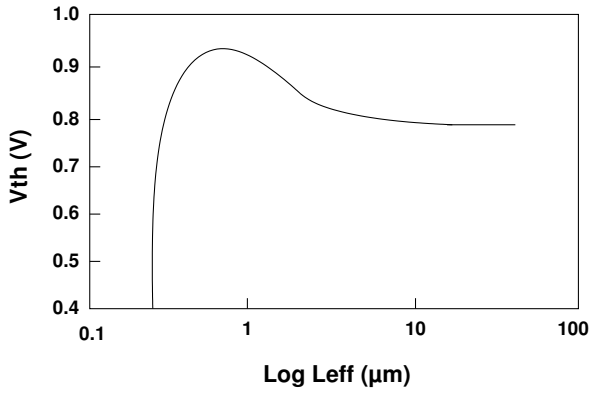

Figure A.2: *Profile of V_{th} versus L_{eff} for reverse short-channel effects.*

A.3 Normal Narrow-Width Effects

Device width has been found to have a significant effect on V_{th}. Under the field oxide, a field implantation may be performed to prevent surface inversion by the gate electrode. This process is known as *Localized oxidation of silicon (LOCOS)*. For the devices fabricated with the widely used LOCOS isolation process, it has been found that V_{th} increases as the channel width decreases as shown in Fig. A.3. This is considered *the normal narrow width effect* and is explained by the contribution of charges in the depletion layer region or in the edge of the filed implant region. As

the width increases, this contribution becomes larger, leading to increased V_{th} [Ji83].

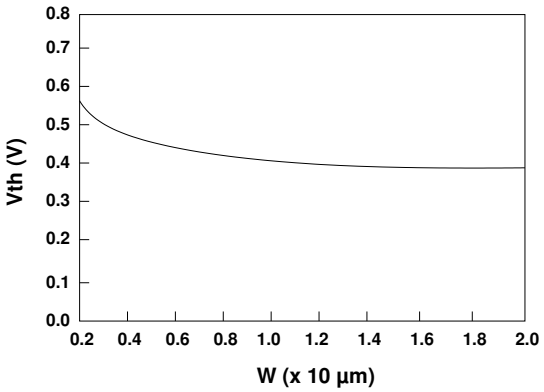

Figure A.3: *Profile of V_{th} versus W for normal narrow-width effects*.

A.4 Reverse Narrow-Width Effects

Another narrow width effect, in which V_{th} decreases as the width channel decreases as shown n Fig. A.4, has been observed in devices with certain new isolating techniques such fully recessed or trench isolation technologies[Hsueh88]. For a device with trench isolation, the field oxide is buried in the substrate and the field lines from the gate electrode are focused by sharp geometry of the channel edge. Thus at the edges of the channel an inversion layer is formed at a lower voltage than at the center. As a result, V_{th} is lower in devices with smaller W's. This behavior is called *the reverse narrow-width effect*, in deference to the "normal" narrow width effect.

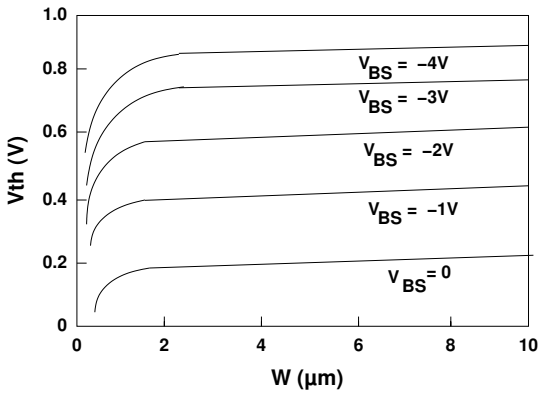

Figure A.4: *Profile of V_{th} versus W for reverse narrow-width effects*.

A.5 Body Bias Effect

The body effect [Cheng99] refers to the influence of V_{bs} on V_{th}. The body effect results in an increase in the threshold voltage of a MOSFET when a reverse bias V_{bs} is applied. This effect is expressed by

$$V_{th} = V_{FB} + 2\phi_B + \gamma\sqrt{2\phi_B - V_{bs}} \tag{A.1}$$

A.6 Bulk charge Effect

The bulk charge effect [Cheng99] is closely related to the body bias effect and refers to the changing threshold voltage along the channel when $V_{ds} > 0$. V_{th} is not constant along the channel because the width is of the depletion region along the channel is not uniform in the presence of a nonzero V_{ds}. In that case, V_{th} will be a function of the position, that is, a function of $V(y)$ along the channel, where $V(y)$ is the channel potential voltage. This is expressed by,

$$V_{th}(y) = V_{th}(0) + \gamma(\sqrt{\phi_B - V_{bs} + V(y)} - \sqrt{\phi_B - V_{bs}}) \tag{A.2}$$

where $V_{th}(0)$ is the threshold voltage at the source and y is the distance from the source.

Appendix B

Device API

In the following sections, our proposed *Application Program Interface* (API) for a device will be described. These API are integrated to the CAIRO+ framework and are called only inside a device to configure it properly for the task of sizing and biasing.

B.0.1 Declaring the Reference Transistor

To declare the reference transistor inside a device, the API

CAIRO_SET_DEVICE_REFERENCE_LOTRS("<tr-name>")

is called during the creation phase of the device by passing to it the name of the reference transistor <tr-name>.

B.0.2 Adding Device Constraints

To add an intrinsic constraint inside a device, the API

CAIRO_ADD_CONSTRAINT("<cnstr-expression>")

is called inside the device SIZE procedure.

The API gets a constraint expression <cnstr-expression> that is built using the macros defined in Table B.1.

B.0.3 Synthesizing the Device

In order to size and bias the device, the API

CAIRO_AUTO_SIZE_AND_BIAS()

is called during the synthesis phase of a device. The API implements the SYNTHESIZE routine, shown in Fig. B.1. For a device, lines 5-6 are the most relevant steps. Line 5 simply says that if the underlying part of the circuit is a device, then in line 6, the dependencies are generated for the reference transistor. The generated dependencies will depend on the interconnection configuration of the reference transistor.

Macro	**Description**
IDS(”<tr-name>”)	The drain-source current of the transistor whose name is <tr-name>, e.g. IDS(”M1”). This macro is used anywhere in an expression.
W1(”<tr-name>”)	The width of the transistor whose name is <tr-name>, e.g. W1(”M1”). This macro is used anywhere in an expression.
W2(<constant-weight>,”<tr-name>”)	The weighted width of the transistor whose name is <tr-name>, e.g. W2(2.0,”M1”). The weight is a floating-point constant. This macro should be the last one used in an expression.
PARAM1(”<param-name>”)	Device parameter whose name is <param-name>. This macro should be the last one used in an expression, e.g. PARAM1(”IDS”).
PARAMD(”<param-name>”)	Device parameter whose name is <param-name>. This macro is used anywhere in an expression, e.g. PARAMD(”IDS”).
PARAM2(<constant-weight>,”<param-name>”)	Weighted device parameter whose name is <tr-name>, e.g. PARAM2(2.0,”IDS”). The weight is a floating-point constant. This macro should be the last one used in an expression.
PARAM2V(<variable-weight>,”<param-name>”)	Weighted device parameter whose name is <tr-name>, e.g. PARAM2V(var,”IDS”). The weight is a floating-point variable. This macro should be the last one used in an expression.
PARAMIO(”<tr-name>”,”<param-name>”)	Parameter <param-name> of transistor named <tr-name>, e.g. PARAMIO(”M1”,”IDS”). This macro is used anywhere in an expression.
EQ, EQUAL	Mathematical equality operators.

Table B.1: *Macros definition for adding intrinsic device constraints*.

```
1     function synthesize( generator )
2         for every child of generator
3             call synthesize(child)
4         end for
5         if generator is a device
6             generate dependencies for the reference transistor
7             eliminate all redundant dependencies
8         else if generator is a module
9             merge dependencies of all children generators
10            eliminate all redundant dependencies
11            if generator is the root generator then
12                resolve all external conflicts
13            end if
14        end if
15    end function
```

Figure B.1: *Pseudo-code of the* SYNTHESIZE *routine*.

Appendix C

Device Implementation

C.1 CREATE procedure

In CAIRO+, each device has its own CREATE procedure. Inside the CREATE procedure, the device declares which transistor is the reference transistor. The device also declares the sizing and biasing operators that are needed to size and bias its reference transistor depending on its interconnection. Finally, the device declares a SIZE procedure that creates the design plan for the device. An example for the CREATE procedure of a differential pair is shown in Fig. C.1.

```
CAIRO_BEGIN_CREATE(DP, char * trName1, char *trName2, ... )

        // Declare Operators
        CAIRO_DECLARE_PROCEDURE("OPVS(VEG)", ... ) ;
        . . .

        // Declare SIZE procedure
        CAIRO_DECLARE_PROCEDURE("SIZE", ... ) ;
        . . .

        // Declare Reference Transistor
        CAIRO_SET_DEVICE_REFERENCE_LOTRS(trName1) ;
        . . .

CAIRO_END_CREATE(DP)
```

Figure C.1: *Example of the implementation of the CREATE procedure of a differential pair.*

C.2 SIZE procedure

In CAIRO+, each device has its own SIZE procedure that is declared in the CREATE procedure. Inside the SIZE procedure, intrinsic device constraints are specified and the SYNTHESIZE routine, shown in Fig. C.3, is called to build the sizing procedure of the device based on the specified design parameters. An example for the SIZE procedure of a differential pair is shown in Fig. C.2. In this procedure, the parameter V_{eg} is checked if it was specified for the device from a higher module level. If specified, the constraint $V_{eg,M_1} = V_{eg,M_2} = V_{eg,DP}$ is added to the device. Then the SYNTHESIZE routine is called via *CAIRO_AUTO_SIZE_AND_BIAS()* to generate the sizing procedure for the device.

```
CAIRO_BEGIN_PROCEDURE("SIZE")
        double veg;
        . . .

        // Add Intrinsic Constraints to the Device
        CAIRO_TRY_GET_VALUE("VEG",veg)
            string expr = PARAMIO(trName1,"VEG") EQ PARAMIO(trName2,"VEG") EQ PARAM1("VEG") ;
            CAIRO_ADD_CONSTRAINT(expr.c_str()) ;
        IF_NO_VALUE
        ENDIF_NO_VALUE
        . . .

        // Generate Sizing Procedure of the Device
        CAIRO_AUTO_SIZE_AND_BIAS();
        . . .

END_PROCEDURE
```

Figure C.2: *Example of the implementation of the SIZE procedure of a differential pair.*

C.3 The SYNTHESIZE routine

The SYNTHESIZE routine is outlined in Fig. C.3. The SYNTHESIZE routine is called via *CAIRO_AUTO_SIZE_AND_BIAS()* to generate the sizing procedure for the device. When called, the steps depicted at lines 2-7 are executed:

1. In lines 2-4, the routine is called recursively for all children generators
2. In line 5, the generator is checked if it represents a device level.

```
1     function synthesize( generator )
2         for every child of generator
3             call synthesize(child)
4         end for
5         if generator is a device
6             generate dependencies for the reference transistor
7             eliminate all redundant dependencies
8         else if generator is a module
9             merge dependencies of all children generators
10            eliminate all redundant dependencies
11            if generator is the root generator then
12                resolve all external conflicts
13            end if
14        end if
15    end function
```

Figure C.3: *Pseudo-code of the* SYNTHESIZE *routine*.

3. In line 6, the reference transistor dependencies are generated as depicted in subsection 6.4.2.

4. In line 7, the redundant dependencies are eliminated in devices as explained in appendix D.6.

Appendix D

CAIRO+: A Dependency Language for Modeling and Design

It is essential for the proposed methodology to express knowledge. We propose to express knowledge using constraints, operators and designer-defined procedures. More important we show how the language joins expressed knowledge to create a reusable, consistent, and top-down design plan. Our aim is to create design plans respecting circuit structure and designer's hypotheses by construction.

The API described in this appendix existed initially in CAIRO+ language[Tuong06]. Our proposed work focused on augmenting the semantics of the API to deal with dependency graphs.

D.1 Capturing module input parameter using *GET_VALUE*

A module input parameter is captured in the current module level by issuing the API function:

CAIRO_GET_VALUE("<mod-param>", <link-var>)

The *GET_VALUE* will create the dependency rule:

<link-var> $\xleftarrow{1.0}$ (<cur-mod-name>/<mod-param>)

which states that the parameter <mod-param> of the current module named <cur-mod-name> will affect the *link* variable defined by the designer. An example code and its corresponding dependency graph are shown in Fig. D.1 and Fig. D.2 respectively.

D.2 Setting a device input parameter using *SET_PARAM*

Once a module parameter has been read into a *link* variable, it can affect a proper device parameter by setting it into the device using *SET_PARAM* which has the syntax

CAIRO_SET_PARAM("<dev-name>","<param-name>",<link-var>)

where <dev-name> is the device name in the next lower level, <param-name> is the parameter name of the device and <link-var> is the link variable that holds the proper value. In this case,

the *SET_PARAM* will create the dependency rule:

(<dev-name>/<param-name>,<dev-name>/<con-name>) $\xleftarrow{1.0}$ <link-var>

which states that the link variable will directly affect the named device parameter that is referring to the external device connector <dev-name>/<con-name> connected to the reference transistor terminals. To illustrate a device, the differential pair device and its external connectors are shown in Fig. D.3 with the gate connection in bold line. In addition, an example code using the differential pair and its corresponding dependency graph are shown in Fig. D.4 and Fig. D.5 respectively.

D.3 Declaring and defining designer-defined procedures (DDP)

A design can express his own knowledge using designer-defined procedure (DDP). In DDP, several input parameters u_1, u_2, u_3, .., u_n are permitted but only one output parameter v is allowed. The DDP expresses the dependence of the single output parameter on the several input parameters:

$$v \xleftarrow{1.0} u_1, u_2, u_3, .., u_n$$

An example code for declaring and defining a DDP is illustrated in Fig. D.6. In the CREATE section, the DDP procedure names *I(K,IBIAS)* is declared. It has *K* and*IBIAS* as input parameters, and *Q* as the output parameter. The DDP procedure is defined in a special section named the *Design Space Exploration*(DSES) section. It reads the two input parameters *K* and *IBIAS* from the higher module level using *GET_VALUE* and computes the output parameter *Q* as the multiplication of *K* and *IBIAS*. The result is set back to the higher level using a *SET_VALUE*.

```
double vin_cm;
. . .
// Capture Common-Mode Input Parameter VIN+ into vin_cm
CAIRO_TRY_GET_VALUE("VIN+", vin_cm);
. . .
```

Figure D.1: *An example code of* GET_VALUE.

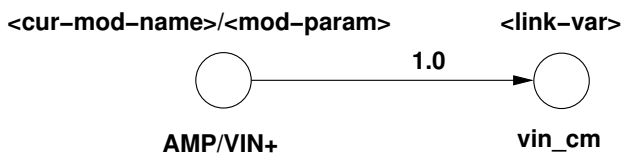

Figure D.2: *Dependency graph representation for* GET_VALUE.

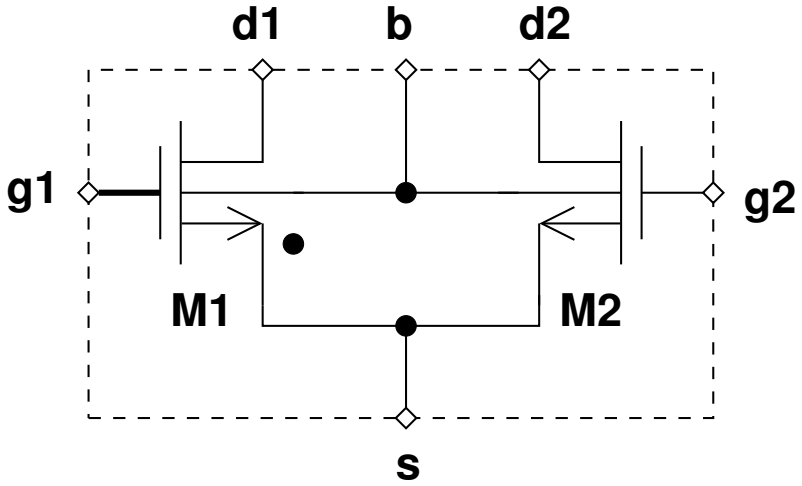

Figure D.3: *External connectors for a differential pair*.

```
double vin_cm;
. . .
// Set gate voltage of the differential pair DP
CAIRO_SET_PARAM("DP", "VG", vin_cm);
. . .
```

Figure D.4: *An example code of* SET_PARAM.

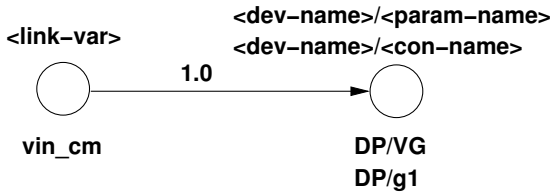

Figure D.5: *Dependency graph representation for* SET_PARAM.

D.4 Retrieving an output parameter from designer-defined procedures (DDP) using *GET_PARAM*

A designer executes the DDP procedure only inside the current module level. Once evaluated the output parameter is read using *GET_PARAM*. An example code is shown in Fig. D.8 . Inside the SYNTHESIZE procedure, the floating-point variable *ids_2nd_stage* is declared. Then the DDP procedure is called using *COMPUTE*. After DDP computation, the output parameter$Q$ is read in the variable *ids_2nd_stage*.

```
CAIRO_BEGIN_CREATE(AMPLIFIER, ... )
        . . .
        // Declare Designer-Defined Procedure called "I(K,IBIAS)"
        // Having K and IBIAS As Inputs and Q As The Only Output
        CAIRO_DECLARE_PROCEDURE("I(K,IBIAS)", .... , "K", CP_IN, "IBIAS", CP_IN, "Q", CP_OUT);
        . . .
CAIRO_END_CREATE(AMPLIFIER)

CAIRO_BEGIN_DSES(AMPLIFIER, ... )

        CAIRO_BEGIN_PROCEDURE("I(K,IBIAS)")
            double k;
            double ibias;
            double result;

            CAIRO_TRY_GET_VALUE("K",k)
            IF_NO_VALUE
                FATAL_ERROR_PARAM("K","K not set in AMPLIFIER",LOCATION) ;
            ENDIF_NO_VALUE

            CAIRO_TRY_GET_VALUE("IBIAS",ibias)
            IF_NO_VALUE
                FATAL_ERROR_PARAM("IBIAS","IBIAS not set in AMPLIFIER",LOCATION) ;
            ENDIF_NO_VALUE

            result = k * ibias;

            CAIRO_SET_VALUE("Q", result, CP_VALID) ;

            CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

CAIRO_END_DSES(AMPLIFIER)
```

Figure D.6: *An example code for declaring and defining DDP*.

When using a DDP procedure, *GET_PARAM* will deduce that the link variable used in the module level depends on the input parameters of the DDP procedure. This will create the dependency:

<link-var> $\xleftarrow{1.0}$ (<cur-mod-name>/<mod-param1>, <cur-mod-name>/<mod-param2>, $\cdots$)

In the example code shown in Fig. D.8, the link variable *ids_2nd_stage* depends on the two input parameters *K* and*IBIAS* of the DDP procedure. The corresponding dependency graph is shown in Fig. D.7.

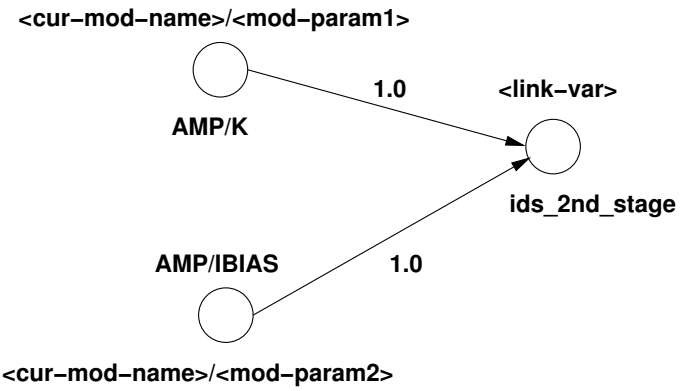

Figure D.7: *Dependency graph generated using DDP mechanism*.

As discussed in section D.1, one dependency rules that corresponds to a *GET_VALUE* inside the DDP will be created. In the case of DDP, this dependency is considered as redundant. Therefore it will be ignored and eliminated from the dependency graph.

D.5 Using *GET_PARAM* inside designer-defined procedures (DPP)

Inside a DDP procedure, the designer may use *GET_PARAM* to retrieve a device parameter. An example is shown in Fig. D.9, where device parameters V_{eg,M_8} and V_{D,M_8} are used to compute V_{D,M_6}.

Inside the SYNTHESIZE procedure, the floating-point variable *vd_m6* is declared. Then the DDP procedure is called using *COMPUTE*. After DDP computation, the module output parameter *Q* is read in the variable *vd_m6*.

When using *GET_PARAM* inside the DDP procedure, *GET_PARAM* will deduce that the link variable used in the module level depends on the device parameters. This will create the dependencies:

<cur-mod-name>/<mod-param>$\xleftarrow{1.0}$(<cur-mod-name> _ <dev1-name>/<dev1-param>, $\cdots$)

<link-var> $\xleftarrow{1.0}$ <cur-mod-name>/<mod-param>

In the example code shown in Fig. D.9, the link variable *vd_m6* depends on the two device parameters V_{eg,M_8} and V_{D,M_8}. The corresponding dependency graph is shown in Fig. D.10.

```
CAIRO_BEGIN_CREATE(AMPLIFIER, ... )
        . . .
        // Declare Designer-Defined Procedure called "I(K,IBIAS)"
        // Having K and IBIAS As Inputs and Q As The Only Output
        CAIRO_DECLARE_PROCEDURE("I(K,IBIAS)", ... , "K", CP_IN, "IBIAS", CP_IN, "Q", CP_OUT);
        . . .
        // Declare SYNTHESIZE procedure
        CAIRO_DECLARE_PROCEDURE("synthesize", ... );
        . . .
CAIRO_END_CREATE(AMPLIFIER)

CAIRO_BEGIN_DSES(AMPLIFIER, ... )

        CAIRO_BEGIN_PROCEDURE("synthesize")
            . . .
            double ids_2nd_stage = 0.0;
            CAIRO_COMPUTE("I(K,IBIAS)");
            CAIRO_GET_PARAM("Q",ids_2nd_stage);
            . . .
        END_PROCEDURE

        CAIRO_BEGIN_PROCEDURE("I(K,IBIAS)")
            double k;
            double ibias;
            double result;

            CAIRO_TRY_GET_VALUE("K",k)
            IF_NO_VALUE
                FATAL_ERROR_PARAM("K","K not set in AMPLIFIER",LOCATION) ;
            ENDIF_NO_VALUE

            CAIRO_TRY_GET_VALUE("IBIAS",ibias)
            IF_NO_VALUE
                FATAL_ERROR_PARAM("IBIAS","IBIAS not set in AMPLIFIER",LOCATION) ;
            ENDIF_NO_VALUE

            result = k * ibias;

            CAIRO_SET_VALUE("Q", result, CP_VALID) ;

            CAIRO_SET_RETURN_PROCEDURE(CP_OK);
        END_PROCEDURE

CAIRO_END_DSES(AMPLIFIER)
```

Figure D.8: *An example code for retrieving a DDP parameter using* GET_PARAM.

```
CAIRO_BEGIN_CREATE(AMPLIFIER, ... )
        . . .
        // Declare Designer-Defined Procedure called "VD_M6BP"
        // Having Q As The Only Output
        CAIRO_DECLARE_PROCEDURE("VD_M6", ... , "Q", CP_OUT );
        . . .
        // Declare SYNTHESIZE procedure
        CAIRO_DECLARE_PROCEDURE("synthesize", ... );
        . . .
CAIRO_END_CREATE(AMPLIFIER)

CAIRO_BEGIN_DSES(AMPLIFIER, ... )

        CAIRO_BEGIN_PROCEDURE("synthesize")
            . . .
            double vd_m6 = 0.0;
            CAIRO_COMPUTE("VD_M6");
            CAIRO_GET_PARAM("Q",vd_m6);
            . . .
        END_PROCEDURE

        CAIRO_BEGIN_PROCEDURE("VD_M6BP")
            double veg_m8;
            double vd_m8;
            double result;

            CAIRO_GET_PARAM("M8","VEG",veg_m8);
            CAIRO_GET_PARAM("L8","VD",vd_m8);

            result = vd_m8 - veg_m8 + 0.1 ;

            CAIRO_SET_VALUE("Q", result, CP_VALID) ;

            CAIRO_SET_RETURN_PROCEDURE(CP_OK);
        END_PROCEDURE

CAIRO_END_DSES(AMPLIFIER)
```

Figure D.9: *An example code for using* GET_PARAM *inside a DDP*.

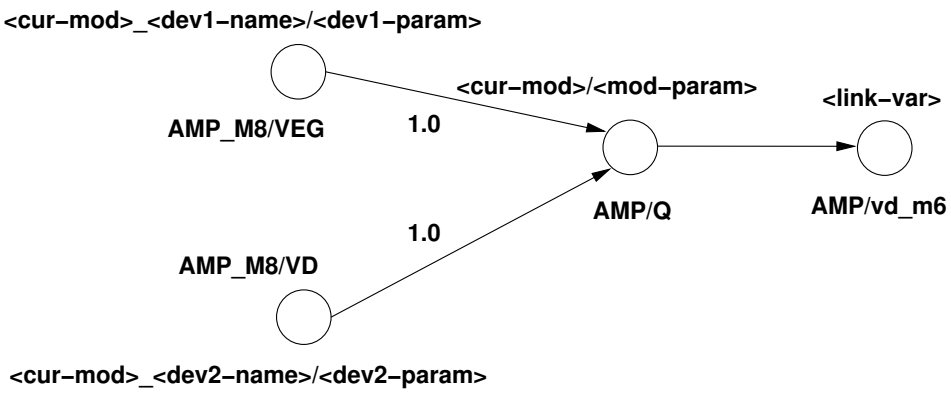

Figure D.10: *Dependency graph generated for DDP using* GET_PARAM.

D.6 Elimination of Redundant Dependencies in Devices

Initially, CAIRO+ language was not intended to be a dependency language for circuit modeling and design. Therefore, some mechanisms had to be implemented in order to be able to express dependencies. As a side effect, some redundancies had to be generated and then filtered. Looking at appendix C.2, one may notice that *GET_VALUE* was used inside the SYNTHESIZE procedure. The syntax of the *GET_VALUE* is:

CAIRO_TRY_GET_VALUE("<param-name>", <link-var>)

The *GET_VALUE* will create the dependency rule:

<link-var> $\xleftarrow{1.0}$ (<cur-dev-name>/<param-name>)

which states that the parameter <param-name> of the current device named <cur-dev-name> will affect the *link* variable defined by the device designer. This is a redundant rule that results from the common semantics of *GET_PARAM* for devices and modules. Therefore, it should be eliminated and removed. The detection of this type of rule is performed as follows: *every rule that a link node that is referenced only once as an affected node and never as an affecting node, should be eliminated*. After generating the dependencies of the reference transistor of a device, the redundant rules has to be detected and eliminated from the pool of dependency rules of a device.

D.7 Elimination of Redundant Dependencies in Modules

Similar to devices, redundant dependency rules has to be eliminated from the pool of dependency rules of a module. Several types of redundant rules are distinguished:

1. A dependency rule that has no affecting nodes is created for each equipotential node, for debugging and tracing purposes. These redundant rules has to be eliminated.

2. Another example of redundant dependencies was mentioned in D.6. This type of redundant rules is created as the semantics of *GET_VALUE* created them. Since these rules do not

impose dependencies for a DPP, those should be eliminated. The detection of such rules is performed in as follows: *every rule that a link node that is referenced only once as an affected node and never as an affecting node, should be eliminated*.

3. A more important class of redundant rules can be eliminated in order to make the design plan independent from the order used to synthesize devices. This type of redundancy has been explained in further details in section 6.4.4.

Appendix E

Module Implementation

E.1 CREATE procedure

In CAIRO+, each root module has its own CREATE procedure. Inside the CREATE procedure, devices and lower-level modules are instantiated and interconnected to form the target topology. The root module also declares the SIZE procedure that is required to synthesize the whole module by synthesizing each child device and lower-level module consisting the root module. The designer-defined procedure is also declared in the CREATE procedure. A detailed example of the CREATE procedure is given for the OTA amplifier in appendices F and G.

E.2 SIZE procedure

In the SIZE procedure, the following essential steps are performed:

1. The equipotentials are identified and preserved.
2. A synthesis section is started.
3. Module input parameters are captured using GET_VALUE.
4. For each device or lower-level module:
   (a) Extrinsic device constraints are declared, if any.
   (b) Some input parameters are computed using designer-defined procedures.
   (c) All the known input parameters are set in every child device and lower-level module.
   (d) The SIZE procedure of every child device and lower-level module, is called in order to generate the corresponding dependency graph
5. Extrinsic module constraints are declared, if any.
6. Newton-Raphson constraints are declared, if any.

7. The SIZE procedure calls the SYNTHESIZE routine to generate the root module dependency graph.
8. The module dependency graph is displayed.
9. The synthesis section is ended.
10. The module dependency graph is evaluated.
11. The module dependency rules are displayed.
12. The output module parameters are set back to the caller.

A detailed example of the SIZE procedure is given for the OTA amplifier in appendices F and G.

Appendix F

The OTA Amplifier CAIRO+ Generator for Designer Mode

```
/*****************************************************************************/
/*                    File : ota_designer_mode.cpp                           */
/*                                                                           */
/* Description : Two-stage Operational Transconductance Amplifier            */
/*                                                                           */
/* Language    : C/C++ et CAIRO+       Version : 1.0                         */
/*                                                                           */
/* Author      : Ramy ISKANDER                                               */
/*                                                                           */
/* Licence     : QPL                                                         */
/*                                                                           */
/* History     :                                                             */
/*                                                                           */
/* Function    : Synthesis in designer mode                                  */
/*                                                                           */
/*****************************************************************************/

#include "cairoplus.h"

/*****************************************************************************/
/* Function: CREATE                                                          */
/*****************************************************************************/
/*                                                                           */
/* This is the CREATE section of the CAIRO+ generator                        */
/*                                                                           */
/*****************************************************************************/

CAIRO_BEGIN_CREATE(OTAS2ET, char *name, char type, bool bulk)

        /**********************\
        |    Initialization     |
        \**********************/

        if (type != TRANSN)
        {
                cerr << "Error: Only OTA2ET/IREF of type TRANSN is allowed." << endl;
                exit(0);
        }

        // ---------------------------------------------------------------
        // Save generator options in local variables
        // ---------------------------------------------------------------

        CAIRO_SET_LOCAL_VARIABLE("TYPE", type);
        CAIRO_SET_LOCAL_VARIABLE("BULK", bulk);

        // -----------------
        // Default Values
```

```
// -----------------

CAIRO_SET_LOCAL_VARIABLE("TEMP", 300.15);

/**********************\
|       Connectors      |
\**********************/

// -------
// Inputs
// -------

CAIRO_IO("VEP", CP_WEST);
CAIRO_IO("VEN", CP_WEST);
CAIRO_IO("VDDT", CP_NORTH);
CAIRO_IO("VSST", CP_SOUTH);

// --------
// Outputs
// --------

CAIRO_IO("VOUT", CP_EAST);

/************************************\
|            NETLIST TEMPLATE          |
\************************************/

// ------------------------------
// Instantiation of Devices
// ------------------------------

CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m8", "tr_m8", TRANSN, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m5", "tr_m5", TRANSN, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "DP_CC" , "dp_m1_m2", "tr_m1", "tr_m2", TRANSN, true,true,!bulk,1,false);
CAIRO_CREATE ("libTRANSISTOR", "CM_ID" , "cm_m3_m4", "tr_m3", "tr_m4", TRANSP, true,true,!bulk,1);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m6", "tr_m6", TRANSP, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m7", "tr_m7", TRANSN, true,true,false,true);

// ------------------------------
// Netlist Connectivity
// ------------------------------

CAIRO_IMPLICIT_CONNECT("tr_m8" , "nd8", "nd8", "VSST" );
CAIRO_IMPLICIT_CONNECT("tr_m5" , "nd5", "nd8", "VSST" );

CAIRO_IMPLICIT_CONNECT("tr_m6" , "VOUT", "nd2", "VDDT" );
CAIRO_IMPLICIT_CONNECT("tr_m7" , "VOUT", "nd8", "VSST" );

if (bulk)
{
       CAIRO_IMPLICIT_CONNECT("dp_m1_m2", "nd1", "nd2", "VEN", "VEP", "nd5","VSST");
       CAIRO_IMPLICIT_CONNECT("cm_m3_m4", "nd1", "nd2", "VDDT", "VDDT" );
}
else
{
       CAIRO_IMPLICIT_CONNECT("dp_m1_m2", "nd1", "nd2", "VEP", "VEN", "nd5");
       CAIRO_IMPLICIT_CONNECT("cm_m3_m4", "nd1", "nd2", "VDDT" );
}

/************************************\
|            LAYOUT TEMPLATE           |
\************************************/

CAIRO_HORIZONTAL_CONTAINER("OTAS2ET_V1" , "cm_m3_m4", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "tr_m6"   , NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_HORIZONTAL_CONTAINER("OTAS2ET_V2" ,"tr_m8", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        ,"tr_m5", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        ,"tr_m7", NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_VERTICAL_CONTAINER(name           ,"OTAS2ET_V2", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        ,"dp_m1_m2",   NOSYM, 0.0, 0.0, 0.0, 0.0
                                        ,"OTAS2ET_V1", NOSYM, 0.0, 0.0, 0.0, 0.0 );

/************************************\
```

```
|      DECLARATION OF PARAMETERS       |
\*************************************/

// -------------------
// Input Parameters
// -------------------

CAIRO_DECLARE_PARAM("TEMP") ;      // Temperature
CAIRO_DECLARE_PARAM("VDD") ;       // Positive supply
CAIRO_DECLARE_PARAM("VSS") ;       // Negative supply
CAIRO_DECLARE_PARAM("L_BIAS") ;    // Length
CAIRO_DECLARE_PARAM("L_DP") ;      // Length
CAIRO_DECLARE_PARAM("L_CM") ;      // Length
CAIRO_DECLARE_PARAM("L_M6") ;      // Length
CAIRO_DECLARE_PARAM("VSP") ;       // Common-mode output voltage
CAIRO_DECLARE_PARAM("VEG_DP") ;    // Overdrive-gate voltage
CAIRO_DECLARE_PARAM("VEG_CM") ;    // Overdrive-gate voltage
CAIRO_DECLARE_PARAM("VEG_BIAS") ;  // Overdrive-gate voltage
CAIRO_DECLARE_PARAM("VEG_M6") ;    // Overdrive-gate voltage
CAIRO_DECLARE_PARAM("VMC") ;       // Common-mode input voltage
CAIRO_DECLARE_PARAM("IBIAS") ;     // Biasing current
CAIRO_DECLARE_PARAM("IREF") ;      // Reference current
CAIRO_DECLARE_PARAM("K") ;         // Factor of currents between the two stages
 CAIRO_DECLARE_PARAM("CCAP") ;      // Compensation capacitance
CAIRO_DECLARE_PARAM("CL") ;        // Load capacitance
CAIRO_DECLARE_PARAM("RL") ;        // Load resistance

// -------------------
// Output Parameters
// -------------------

CAIRO_DECLARE_PARAM("VBIAS") ;
CAIRO_DECLARE_PARAM("Q") ;

// ---------------------------------
// Primary Performance Parameters
// ---------------------------------

CAIRO_DECLARE_PARAM("AD0") ;            // Static differential-mode gain
CAIRO_DECLARE_PARAM("AC0") ;            // Static common-mode gain
CAIRO_DECLARE_PARAM("FT") ;             // Transition frequency
CAIRO_DECLARE_PARAM("PM") ;             // Phase margin
CAIRO_DECLARE_PARAM("POWER") ;          // Power consumption
CAIRO_DECLARE_PARAM("AREA") ;           // Area
CAIRO_DECLARE_PARAM("SR");              // Slew rate
CAIRO_DECLARE_PARAM("SNE_1HZ");         // Input-referred noise @ 1HZ
CAIRO_DECLARE_PARAM("SNE_FT");          // Input-referred noise @ FT
CAIRO_DECLARE_PARAM("SATURATIONS");     // Transistors in saturation
CAIRO_DECLARE_PARAM("ED0");             // Systematic input offset

// ---------------------------------
// Secondary Performance Parameters
// ---------------------------------

CAIRO_DECLARE_PARAM("CMRR");            // Common-mode rejection ratio
CAIRO_DECLARE_PARAM("COUT");            // Output capacitance
CAIRO_DECLARE_PARAM("FPND") ;           // First non-dominant pole
CAIRO_DECLARE_PARAM("FPD") ;            // First dominant pole
CAIRO_DECLARE_PARAM("FZ") ;             // First zero
CAIRO_DECLARE_PARAM("CIN-");            // Negative-input capacitance
CAIRO_DECLARE_PARAM("CIN+");            // Positive-input capacitance
CAIRO_DECLARE_PARAM("VICMMAX");         // Maximum input common-mode voltage
CAIRO_DECLARE_PARAM("VICMMIN");         // Minimum input common-mode voltage
CAIRO_DECLARE_PARAM("VOUTMAX");         // Maximum output voltage
CAIRO_DECLARE_PARAM("VOUTMIN");         // Minimum output voltage

/*************************************\
|       DECLARATION OF PROCEDURES      |
\*************************************/

// DESIGNER MODE
CAIRO_DECLARE_PROCEDURE("synthesize_with_offset", CP_COMPLETE,  "TEMP",     CP_IN,
                                                                "VDD",      CP_IN,
                                                                "VSS",      CP_IN,
```

```
                                                    "L_M6",     CP_IN,
                                                    "L_DP",     CP_IN,
                                                    "L_CM",     CP_IN,
                                                    "L_BIAS",   CP_IN,
                                                    "VEG_BIAS", CP_IN,
                                                    "VEG_CM",   CP_IN,
                                                    "VEG_DP",   CP_IN,
                                                    "VEG_M6",   CP_IN,
                                                    "VMC",      CP_IN,
                                                    "VSP",      CP_IN,
                                                    "IBIAS",    CP_IN,
                                                    "K",        CP_IN,
                                                    "VBIAS",    CP_OUT ) ;

// HELPERS
CAIRO_DECLARE_PROCEDURE("I(K,IBIAS)"     ,CP_UNCOMPLETE , "K", CP_IN, "IBIAS", CP_IN, "Q", CP_OUT);
CAIRO_DECLARE_PROCEDURE("NI(K,IBIAS)"    ,CP_UNCOMPLETE , "K", CP_IN, "IBIAS", CP_IN, "Q", CP_OUT );
CAIRO_DECLARE_PROCEDURE("IDS_CM(IBIAS)"  ,CP_UNCOMPLETE , "IBIAS", CP_IN, "Q", CP_OUT );
CAIRO_DECLARE_PROCEDURE("IDS_DP(IBIAS)"  ,CP_UNCOMPLETE , "IBIAS", CP_IN, "Q", CP_OUT );

// DISPLAY
CAIRO_DECLARE_PROCEDURE("DISPLAY"  ,CP_UNCOMPLETE );

// EXPLORER
CAIRO_DECLARE_PROCEDURE("explorer"  ,CP_COMPLETE );

// PERFORMANCES
CAIRO_DECLARE_PROCEDURE("PERFORMANCES" ,CP_UNCOMPLETE, "TEMP",        CP_IN,
                                                       "VDD",         CP_IN,
                                                       "VSS",         CP_IN,
                                                       "L_DP",        CP_IN,
                                                       "L_M6",        CP_IN,
                                                       "L_CM",        CP_IN,
                                                       "L_BIAS",      CP_IN,
                                                       "IREF",        CP_IN,
                                                       "VEG_BIAS",    CP_IN,
                                                       "VEG_CM",      CP_IN,
                                                       "VEG_DP",      CP_IN,
                                                       "VEG_M6",      CP_IN,
                                                       "VMC",         CP_IN,
                                                       "VSP",         CP_IN,
                                                       "K",           CP_IN,
                                                       "IBIAS",       CP_IN,
                                                       "CCAP",        CP_IN,
                                                       "CL",          CP_IN,
                                                       "RL",          CP_IN,
                                                       "AD0",         CP_OUT,
                                                       "AC0",         CP_OUT,
                                                       "FT",          CP_OUT,
                                                       "PM",          CP_OUT,
                                                       "SR",          CP_OUT,
                                                       "SNE_1HZ",     CP_OUT,
                                                       "SNE_FT",      CP_OUT,
                                                       "SATURATIONS", CP_OUT,
                                                       "ED0",         CP_OUT,
                                                       "POWER",       CP_OUT,
                                                       "AREA",        CP_OUT)

// MEASURES
CAIRO_DECLARE_PROCEDURE("MEASURES" ,CP_UNCOMPLETE , "TEMP",     CP_IN,
                                                    "VDD",      CP_IN,
                                                    "VSS",      CP_IN,
                                                    "L_M6",     CP_IN,
                                                    "L_DP",     CP_IN,
                                                    "L_CM",     CP_IN,
                                                    "L_BIAS",   CP_IN,
                                                    "IREF",     CP_IN,
                                                    "VEG_BIAS", CP_IN,
                                                    "VEG_CM",   CP_IN,
                                                    "VEG_DP",   CP_IN,
                                                    "VEG_M6",   CP_IN,
                                                    "VMC",      CP_IN,
                                                    "VSP",      CP_IN,
                                                    "K",        CP_IN,
                                                    "IBIAS",    CP_IN,
```

```
                                                    "CCAP",     CP_IN,
                                                    "CL",       CP_IN,
                                                    "RL",       CP_IN,
                                                    "FPND",     CP_OUT,
                                                    "FPD",      CP_OUT,
                                                    "FZ",       CP_OUT,
                                                    "VOUTMAX",  CP_OUT,
                                                    "VOUTMIN",  CP_OUT,
                                                    "VICMMAX",  CP_OUT,
                                                    "VICMMIN",  CP_OUT,
                                                    "CIN-",     CP_OUT,
                                                    "CIN+",     CP_OUT,
                                                    "COUT",     CP_OUT,
                                                    "CMRR",     CP_OUT);
CAIRO_END_CREATE(OTAS2ET)

/*************************************************************************/
/* Function: DSES                                                        */
/*************************************************************************/
/*                                                                       */
/* This is the DESIGN SPACE EXPLORATION section of the CAIRO+ generator  */
/*                                                                       */
/*************************************************************************/

CAIRO_BEGIN_DSES(OTAS2ET,char *name)

        /************************************\
        |           SIZING PROCEDURES        |
        \************************************/

           // --------------------------------
          // Procedure: synthesize_with_offset
          // --------------------------------

         CAIRO_BEGIN_PROCEDURE("synthesize_with_offset")

                    double temp;
                    double ibias;
                    double vdd;
                    double vss;
                    double l_m6;
                    double l_dp;
                    double l_cm;
                    double l_bias;
                    double veg_bias;
                    double veg_cm;
                    double veg_dp;
                    double veg_m6;
                    double vmc;
                    double vsp;

                CAIRO_CREATE_EQUIPOTENTIALS;

                     CAIRO_BEGIN_SYNTHESIS

                        /************************************\
                        |  Reading Parameters from higher level |
                        \************************************/

                        CAIRO_TRY_GET_VALUE("TEMP",temp)
                               IF_NO_VALUE
                                FATAL_ERROR_PARAM("TEMP","TEMP not set in OTAS2ET",LOCATION) ;
                        ENDIF_NO_VALUE

                        CAIRO_TRY_GET_VALUE("IBIAS",ibias)
                               IF_NO_VALUE
                                FATAL_ERROR_PARAM("IBIAS","IBIAS not set in OTAS2ET",LOCATION) ;
                        ENDIF_NO_VALUE

                        CAIRO_TRY_GET_VALUE("L_M6",l_m6)
                               IF_NO_VALUE
                                FATAL_ERROR_PARAM("L_M6","L_M6 not set in OTAS2ET",LOCATION) ;
                        ENDIF_NO_VALUE

                        CAIRO_TRY_GET_VALUE("L_DP",l_dp)
```

```
       IF_NO_VALUE
        FATAL_ERROR_PARAM("L_DP","L_DP not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("L_CM",l_cm)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("L_CM","L_CM not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("L_BIAS",l_bias)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("L_BIAS","L_BIAS not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEG_M6",veg_m6)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VEG_M6","VEG_M6 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEG_DP",veg_dp)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VEG_DP","VEG_DP not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEG_CM", veg_cm)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VEG_CM","VEG_CM not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEG_BIAS", veg_bias)
IF_NO_VALUE
        FATAL_ERROR_PARAM("VEG_BIAS","VEG_BIAS not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VMC", vmc)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VMC","VMC not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VSP", vsp)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VSP","VSP not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VDD",vdd)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VDD","V not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VSS",vss)
       IF_NO_VALUE
        FATAL_ERROR_PARAM("VSS","VSS not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

// --------------
// Synthesize CM
// --------------

cout << "Synthesizing CM ..." << endl;

double ids_cm = 0.0;
CAIRO_COMPUTE("IDS_CM(IBIAS)");
CAIRO_GET_PARAM("Q",ids_cm);

CAIRO_SET_PARAM("cm_m3_m4","TEMP",temp) ;
CAIRO_SET_PARAM("cm_m3_m4","VEG",veg_cm) ;
CAIRO_SET_PARAM("cm_m3_m4","L",l_cm) ;
CAIRO_SET_PARAM("cm_m3_m4","IDS",ids_cm) ;
CAIRO_SET_PARAM("cm_m3_m4","VS",vdd) ;

CAIRO_COMPUTE("cm_m3_m4","synthesize");
ON_PROCEDURE_STATUS
        CAIRO_ERROR_MESSAGE("cannot synthesize CM") ;
END_ON_PROCEDURE_STATUS
```

```
// --------------
// Synthesize DP
// --------------

cout << "Synthesizing DP ..." << endl;

double ids_dp = 0.0;
CAIRO_COMPUTE("IDS_DP(IBIAS)");
CAIRO_GET_PARAM("Q",ids_dp);

CAIRO_SET_PARAM("dp_m1_m2","TEMP",temp) ;
CAIRO_SET_PARAM("dp_m1_m2","VEG",veg_dp) ;
CAIRO_SET_PARAM("dp_m1_m2","L",l_dp) ;
CAIRO_SET_PARAM("dp_m1_m2","IDS",ids_dp) ;
CAIRO_SET_PARAM("dp_m1_m2","VG" ,vmc) ;
CAIRO_SET_PARAM("dp_m1_m2","VB",vss) ;

CAIRO_COMPUTE("dp_m1_m2","synthesize");
ON_PROCEDURE_STATUS
        CAIRO_ERROR_MESSAGE("cannot synthesize DP") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR5
// ---------------

cout << "Synthesizing TR5 ..." << endl;

CAIRO_SET_PARAM("tr_m5","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m5","VEG",veg_bias) ;
CAIRO_SET_PARAM("tr_m5","L",l_bias) ;
CAIRO_SET_PARAM("tr_m5","IDS",ibias) ;

CAIRO_COMPUTE("tr_m5","synthesize");
ON_PROCEDURE_STATUS
        CAIRO_ERROR_MESSAGE("cannot synthesize TR5") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR6
// ---------------

cout << "Synthesizing TR6 ..." << endl;

double ids_m6 = 0.0;
CAIRO_COMPUTE("NI(K,IBIAS)");
CAIRO_GET_PARAM("Q",ids_m6);

string expr = DPARAM("tr_m6","VG") EQ DPARAM("dp_m1_m2","VD");
CAIRO_ADD_CONSTRAINT(expr.c_str());

CAIRO_SET_PARAM("tr_m6","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m6","VEG",veg_m6) ;
CAIRO_SET_PARAM("tr_m6","L",l_m6 ) ;
CAIRO_SET_PARAM("tr_m6","IDS",ids_m6 ) ;
CAIRO_SET_PARAM("tr_m6","VD" ,vsp ) ;

CAIRO_COMPUTE("tr_m6","synthesize");
ON_PROCEDURE_STATUS
        CAIRO_ERROR_MESSAGE("cannot synthesize TR6") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR7
// ---------------

cout << "Synthesizing TR7 ..." << endl;

double ids_m7 = 0.0;
CAIRO_COMPUTE("I(K,IBIAS)");
CAIRO_GET_PARAM("Q",ids_m7);

CAIRO_SET_PARAM("tr_m7","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m7","L"  ,l_bias) ;
CAIRO_SET_PARAM("tr_m7","IDS",ids_m7 ) ;
```

```
        CAIRO_COMPUTE("tr_m7","synthesize");
        ON_PROCEDURE_STATUS
                CAIRO_ERROR_MESSAGE("cannot synthesize TR7") ;
        END_ON_PROCEDURE_STATUS

        // ---------------
        // Synthesize TR8
        // ---------------

        cout << "Synthesizing TR8 ..." << endl;

        expr = DPARAM("tr_m8","TEMP") EQ DPARAM("tr_m5","TEMP");
        CAIRO_ADD_CONSTRAINT(expr.c_str());

        expr = DPARAM("tr_m8","W") EQ DPARAM("tr_m5","W");
        CAIRO_ADD_CONSTRAINT(expr.c_str());

        expr = DPARAM("tr_m8","L") EQ DPARAM("tr_m5","L");
        CAIRO_ADD_CONSTRAINT(expr.c_str());

        CAIRO_COMPUTE("tr_m8","synthesize");
        ON_PROCEDURE_STATUS
                CAIRO_ERROR_MESSAGE("cannot synthesize TR8") ;
        END_ON_PROCEDURE_STATUS

        // -------------------
        // Synthesize device
        // -------------------

        CAIRO_AUTO_SIZE_AND_BIAS();

        // -------------------
        // Display Graphs
        // -------------------

         CAIRO_DISPLAY_GRAPHS();

CAIRO_END_SYNTHESIS

// -------------------
// Execute design plan
// -------------------

CAIRO_EXECUTE_DESIGN_PLAN();

// -------------------
// Display Rules
// -------------------

CAIRO_DISPLAY_RULES();

// ----------------------------
// Call direct size for devices
// ----------------------------

CAIRO_COMPUTE("cm_m3_m4","direct_size");

CAIRO_COMPUTE("dp_m1_m2","direct_size");

CAIRO_COMPUTE("tr_m5","direct_size");

CAIRO_COMPUTE("tr_m6","direct_size");

CAIRO_COMPUTE("tr_m7","direct_size");

CAIRO_COMPUTE("tr_m8","direct_size");

 // ------------------------
 // Return polarization
 // ------------------------

 double vbias = 0.0;

 CAIRO_GET_PARAM("tr_m5","VG"  ,vbias) ;
```

```
        CAIRO_SET_VALUE("VBIAS", vbias, CP_VALID) ;

        CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ---------------------------
// Procedure: IDS_CM(IBIAS)
// ---------------------------

CAIRO_BEGIN_PROCEDURE("IDS_CM(IBIAS)")

       double ibias;

       CAIRO_TRY_GET_VALUE("IBIAS",ibias)
       IF_NO_VALUE
               FATAL_ERROR_PARAM("IBIAS","IBIAS not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       double result = -ibias / 2.0;

       CAIRO_SET_VALUE("Q", result, CP_VALID) ;

       CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ---------------------------
// Procedure: IDS_DP(IBIAS)
// ---------------------------

  CAIRO_BEGIN_PROCEDURE("IDS_DP(IBIAS)")

       double ibias;

       CAIRO_TRY_GET_VALUE("IBIAS",ibias)
       IF_NO_VALUE
               FATAL_ERROR_PARAM("IBIAS","IBIAS not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       double result = ibias / 2.0;

       CAIRO_SET_VALUE("Q", result, CP_VALID) ;

       CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ---------------------------
// Procedure: NI(K,IBIAS)
// ---------------------------

CAIRO_BEGIN_PROCEDURE("NI(K,IBIAS)")

       double k;
       double ibias;

       CAIRO_TRY_GET_VALUE("K",k)
              IF_NO_VALUE
               FATAL_ERROR_PARAM("K","K not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       CAIRO_TRY_GET_VALUE("IBIAS",ibias)
       IF_NO_VALUE
               FATAL_ERROR_PARAM("IBIAS","IBIAS not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       double result = - k * ibias;

       CAIRO_SET_VALUE("Q", result, CP_VALID) ;

       CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE
```

```
// ----------------------------
// Procedure: I(K,IBIAS)
// ----------------------------

CAIRO_BEGIN_PROCEDURE("I(K,IBIAS)")

       double k;
       double ibias;

       CAIRO_TRY_GET_VALUE("K",k)
              IF_NO_VALUE
               FATAL_ERROR_PARAM("K","K not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       CAIRO_TRY_GET_VALUE("IBIAS",ibias)
       IF_NO_VALUE
               FATAL_ERROR_PARAM("IBIAS","IBIAS not set in OTAS2ET",LOCATION) ;
       ENDIF_NO_VALUE

       double result = k * ibias;

       CAIRO_SET_VALUE("Q", result, CP_VALID) ;

       CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ----------------------------
// Procedure: explorer
// ----------------------------

CAIRO_BEGIN_PROCEDURE("explorer")

      CAIRO_BEGIN_EXPLORE

               // ------------------
               // Execute IE
               // ------------------

               CAIRO_IE_USE();

               CAIRO_IE_ADD_INDEPENDENT_PARAMETERS();

               // Performances
               CAIRO_IE_ADD_WATCH("PERFORMANCES","AD0");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","AC0");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","SATURATIONS");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","ED0");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","SNE_1HZ");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","SNE_FT");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","PM");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","SR");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","FT");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","POWER");
               CAIRO_IE_ADD_WATCH("PERFORMANCES","AREA");

               // Measures
               CAIRO_IE_ADD_WATCH("MEASURES","CMRR");
               CAIRO_IE_ADD_WATCH("MEASURES","FPND");
               CAIRO_IE_ADD_WATCH("MEASURES","FPD");
               CAIRO_IE_ADD_WATCH("MEASURES","FZ");
               CAIRO_IE_ADD_WATCH("MEASURES","CIN-");
               CAIRO_IE_ADD_WATCH("MEASURES","CIN+");
               CAIRO_IE_ADD_WATCH("MEASURES","COUT");
               CAIRO_IE_ADD_WATCH("MEASURES","VICMMIN");
               CAIRO_IE_ADD_WATCH("MEASURES","VICMMAX");
               CAIRO_IE_ADD_WATCH("MEASURES","VOUTMIN");
               CAIRO_IE_ADD_WATCH("MEASURES","VOUTMAX");

               // Wrappers
               CAIRO_IE_SET_DISPLAY_WRAPPER("DISPLAY");
               CAIRO_IE_SET_DISPLAY_FUNCTION("ALL(VGS,W,L)");
```

```
                CAIRO_IE_DISPLAY(name);

                CAIRO_IE_RELEASE();

        CAIRO_END_EXPLORE

        if (CAIRO_IE_EXIT == false)
        {
                // -------------------
                // Execute design plan
                // -------------------

                CAIRO_EXECUTE_DESIGN_PLAN();

                // ----------------------------
                // Call direct size for devices
                // ----------------------------

                CAIRO_COMPUTE("cm_m3_m4","direct_size");

                CAIRO_COMPUTE("dp_m1_m2","direct_size");

                CAIRO_COMPUTE("tr_m6","direct_size");

                CAIRO_COMPUTE("tr_m7","direct_size");

                CAIRO_COMPUTE("tr_m5","direct_size");

                CAIRO_COMPUTE("tr_m8","direct_size");
        }

        CAIRO_SET_RETURN_PROCEDURE(CP_OK);

 END_PROCEDURE

 // ------------------------------
 // Procedure: DISPLAY
 // ------------------------------

 CAIRO_BEGIN_PROCEDURE("DISPLAY")

        // ------------------------------------
        // Calculate small signals for DP
        // ------------------------------------

        double vs_dp;
        double vd_dp;
        double vg_dp;
        double vb_dp;

        CAIRO_GET_PARAM("dp_m1_m2","VD",vd_dp);
        CAIRO_GET_PARAM("dp_m1_m2","VS",vs_dp);
        CAIRO_GET_PARAM("dp_m1_m2","VG",vg_dp);
        CAIRO_GET_PARAM("dp_m1_m2","VB",vb_dp);

        CAIRO_SET_PARAM("dp_m1_m2","VDS", vd_dp - vs_dp);
        CAIRO_SET_PARAM("dp_m1_m2","VBS", vb_dp - vs_dp);
        CAIRO_SET_PARAM("dp_m1_m2","VGS", vg_dp - vs_dp);

        CAIRO_SET_PARAM("dp_m1_m2","SIF_FREQ", 1.0);

        CAIRO_COMPUTE("dp_m1_m2","ALL(VGS,W,L)");

        // To repeat similarly for the rest of devices
        . . .

        // ------------------------------------
        // Return Code
        // ------------------------------------

        CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE
```

```
        // ------------------------------
        // Procedure: PERFORMANCES
        // ------------------------------

        CAIRO_BEGIN_PROCEDURE("PERFORMANCES")

                // Compute primary performances here
                . . .

                CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

        // ------------------------------
        // Procedure: MEASURES
        // ------------------------------

        CAIRO_BEGIN_PROCEDURE("MEASURES")

                // Compute secondary performances here
                . . .

                CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

        // ----------------------------
        // ------------ Default
        // ----------------------------

        CAIRO_DEFAULT_PROCEDURE
              FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"unknown procedure",LOCATION) ;
        END_DEFAULT_PROCEDURE

CAIRO_END_DSES(OTAS2ET)

/*****************************************************************************/
/* Function: LAYOUT                                                          */
/*****************************************************************************/
/*                                                                           */
/* This is the ROUTE section of the CAIRO+ generator                         */
/*                                                                           */
/*****************************************************************************/

CAIRO_BEGIN_LAYOUT(OTAS2ET, char *name)

     // Perform procedural routing here
     . . .

CAIRO_END_LAYOUT(OTAS2ET)
```

Appendix G

The OTA Amplifier CAIRO+ Generator for Simulator Mode

```
/*****************************************************************************/
/*                     File : ota_simulator_mode.cpp                         */
/*                                                                           */
/* Description : Two-stage Operational Transconductance Amplifier            */
/*                                                                           */
/* Language    : C/C++ et CAIRO+       Version : 1.0                         */
/*                                                                           */
/* Author      : Ramy ISKANDER                                               */
/*                                                                           */
/* Licence     : QPL                                                         */
/*                                                                           */
/* History     :                                                             */
/*                                                                           */
/* Function    : Synthesis in simulator mode                                 */
/*                                                                           */
/*****************************************************************************/

#include "cairoplus.h"

/*****************************************************************************/
/* Function: CREATE                                                          */
/*****************************************************************************/
/*                                                                           */
/* This is the CREATE section of the CAIRO+ generator                        */
/*                                                                           */
/*****************************************************************************/

CAIRO_BEGIN_CREATE(OTAS2ET, char *name, char type)

        /***********************\
        |    Initialization     |
        \***********************/

        if (type != TRANSN)
        {
                cerr << "Error: Only OTA2ET/IREF of type TRANSN is allowed." << endl;
                exit(0);
        }

        // ---------------------------------------------------------------
        // Save generator options in local variables
        // ---------------------------------------------------------------

        CAIRO_SET_LOCAL_VARIABLE("TYPE", type);

        // -----------------
        // Default Values
        // -----------------
```

```
CAIRO_SET_LOCAL_VARIABLE("TEMP", 300.15);

/**********************\
|       Connectors     |
\**********************/

// -------
// Inputs
// -------

CAIRO_IO("VEPT", CP_WEST);
CAIRO_IO("VENT", CP_WEST);
CAIRO_IO("VDDT", CP_NORTH);
CAIRO_IO("VSST", CP_SOUTH);

// --------
// Outputs
// --------

CAIRO_IO("VSP", CP_EAST);

/************************************\
|           NETLIST TEMPLATE         |
\************************************/

// ------------------------------
// Instantiation of Devices
// ------------------------------

CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m8", "tr_m8", TRANSN, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m5", "tr_m5", TRANSN, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m1", "tr_m1", TRANSN, true,true,false,false);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m2", "tr_m2", TRANSN, true,true,false,false);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m3", "tr_m3", TRANSP, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m4", "tr_m4", TRANSP, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m6", "tr_m6", TRANSP, true,true,false,true);
CAIRO_CREATE ("libTRANSISTOR", "TR_MOS", "tr_m7", "tr_m7", TRANSN, true,true,false,true);

// ------------------------------
// Netlist Connectivity
// ------------------------------

CAIRO_IMPLICIT_CONNECT("tr_m8" , "nd8", "nd8", "VSST" );
CAIRO_IMPLICIT_CONNECT("tr_m5" , "nd5", "nd8", "VSST" );

CAIRO_IMPLICIT_CONNECT("tr_m6" , "VSP", "nd2", "VDDT" );
CAIRO_IMPLICIT_CONNECT("tr_m7" , "VSP", "nd8", "VSST" );

CAIRO_IMPLICIT_CONNECT("tr_m1" , "nd1", "VENT", "nd5", "VSST" );
CAIRO_IMPLICIT_CONNECT("tr_m2" , "nd2", "VEPT", "nd5", "VSST" );

CAIRO_IMPLICIT_CONNECT("tr_m3" , "nd1", "nd1", "VDDT" );
CAIRO_IMPLICIT_CONNECT("tr_m4" , "nd2", "nd1", "VDDT" );

/************************************\
|            LAYOUT TEMPLATE         |
\************************************/

CAIRO_HORIZONTAL_CONTAINER("OTAS2ET_CM" , "tr_m3", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "tr_m4", NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_HORIZONTAL_CONTAINER("OTAS2ET_DP" , "tr_m1", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "tr_m2", NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_HORIZONTAL_CONTAINER("OTAS2ET_H1" , "tr_m8", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "tr_m5", NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_VERTICAL_CONTAINER("OTAS2ET_V1"   , "OTAS2ET_H1", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "OTAS2ET_DP", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "OTAS2ET_CM", NOSYM, 0.0, 0.0, 0.0, 0.0 );

CAIRO_VERTICAL_CONTAINER("OTAS2ET_V2"   , "tr_m7", NOSYM, 0.0, 0.0, 0.0, 0.0
                                        , "tr_m6", NOSYM, 0.0, 0.0, 0.0, 0.0);

CAIRO_HORIZONTAL_CONTAINER(name         , "OTAS2ET_V1", NOSYM, 0.0, 0.0, 0.0, 0.0
```

```
                                    , "OTAS2ET_V2", NOSYM, 0.0, 0.0, 0.0, 0.0);

/************************************\
|      DECLARATION OF PARAMETERS       |
\************************************/

// -------------------
// Input Parameters
// -------------------

CAIRO_DECLARE_PARAM("TEMP") ;      // Temperature
CAIRO_DECLARE_PARAM("VDD") ;       // Positive supply
CAIRO_DECLARE_PARAM("VSS") ;       // Negative supply
CAIRO_DECLARE_PARAM("W_M1") ;      // Width
CAIRO_DECLARE_PARAM("W_M2") ;      // Width
CAIRO_DECLARE_PARAM("W_M3") ;      // Width
CAIRO_DECLARE_PARAM("W_M4") ;      // Width
CAIRO_DECLARE_PARAM("W_M5") ;      // Width
CAIRO_DECLARE_PARAM("W_M6") ;      // Width
CAIRO_DECLARE_PARAM("W_M7") ;      // Width
CAIRO_DECLARE_PARAM("W_M8") ;      // Width
CAIRO_DECLARE_PARAM("L_M1") ;      // Length
CAIRO_DECLARE_PARAM("L_M2") ;      // Length
CAIRO_DECLARE_PARAM("L_M3") ;      // Length
CAIRO_DECLARE_PARAM("L_M4") ;      // Length
CAIRO_DECLARE_PARAM("L_M5") ;      // Length
CAIRO_DECLARE_PARAM("L_M6") ;      // Length
CAIRO_DECLARE_PARAM("L_M7") ;      // Length
CAIRO_DECLARE_PARAM("L_M8") ;      // Length
CAIRO_DECLARE_PARAM("VOUT") ;      // Common-mode output voltage
CAIRO_DECLARE_PARAM("VEN") ;       // Negative terminal common-mode input
CAIRO_DECLARE_PARAM("VEP") ;       // Positive terminal common-mode input
CAIRO_DECLARE_PARAM("IREF") ;      // Reference current
CAIRO_DECLARE_PARAM("CCAP") ;      // Compensation capacitance
CAIRO_DECLARE_PARAM("CL") ;        // Load capacitance
CAIRO_DECLARE_PARAM("RL") ;        // Load resistance

CAIRO_DECLARE_PARAM("VG_M3") ;     // Gate node voltage of M3

// -------------------
// Output Parameters
// -------------------

CAIRO_DECLARE_PARAM("VBIAS") ;
CAIRO_DECLARE_PARAM("Q") ;

// --------------------------------
// Primary Performance Parameters
// --------------------------------

CAIRO_DECLARE_PARAM("AD0") ;            // Static differential-mode gain
CAIRO_DECLARE_PARAM("AC0") ;            // Static common-mode gain
CAIRO_DECLARE_PARAM("FT") ;             // Transition frequency
CAIRO_DECLARE_PARAM("PM") ;             // Phase margin
CAIRO_DECLARE_PARAM("POWER") ;          // Power consumption
CAIRO_DECLARE_PARAM("AREA") ;           // Area
CAIRO_DECLARE_PARAM("SR");              // Slew rate
CAIRO_DECLARE_PARAM("SNE_1HZ");         // Input-referred noise @ 1HZ
CAIRO_DECLARE_PARAM("SNE_FT");          // Input-referred noise @ FT
CAIRO_DECLARE_PARAM("SATURATIONS");     // Transistors in saturation
CAIRO_DECLARE_PARAM("ED0");             // Systematic input offset

// --------------------------------
// Secondary Performance Parameters
// --------------------------------

CAIRO_DECLARE_PARAM("CMRR");            // Common-mode rejection ratio
CAIRO_DECLARE_PARAM("COUT");            // Output capacitance
CAIRO_DECLARE_PARAM("FPND") ;           // First non-dominant pole
CAIRO_DECLARE_PARAM("FPD") ;            // First dominant pole
CAIRO_DECLARE_PARAM("FZ") ;             // First zero
CAIRO_DECLARE_PARAM("CIN-");            // Negative-input capacitance
CAIRO_DECLARE_PARAM("CIN+");            // Positive-input capacitance
CAIRO_DECLARE_PARAM("VICMMAX");         // Maximum input common-mode voltage
CAIRO_DECLARE_PARAM("VICMMIN");         // Minimum input common-mode voltage
```

```
CAIRO_DECLARE_PARAM("VOUTMAX");         // Maximum output voltage
CAIRO_DECLARE_PARAM("VOUTMIN");         // Minimum output voltage

/***********************************\
|       DECLARATION OF PROCEDURES      |
\***********************************/

// SIMULATOR MODE
CAIRO_DECLARE_PROCEDURE("simulate_with_input_offset_and_feedback", CP_COMPLETE, "TEMP",  CP_IN,
                                                                                "VDD",   CP_IN,
                                                                                "VSS",   CP_IN,
                                                                                "W_M1",  CP_IN,
                                                                                "W_M2",  CP_IN,
                                                                                "W_M3",  CP_IN,
                                                                                "W_M4",  CP_IN,
                                                                                "W_M5",  CP_IN,
                                                                                "W_M6",  CP_IN,
                                                                                "W_M7",  CP_IN,
                                                                                "W_M8",  CP_IN,
                                                                                "L_M1",  CP_IN,
                                                                                "L_M2",  CP_IN,
                                                                                "L_M3",  CP_IN,
                                                                                "L_M4",  CP_IN,
                                                                                "L_M5",  CP_IN,
                                                                                "L_M6",  CP_IN,
                                                                                "L_M7",  CP_IN,
                                                                                "L_M8",  CP_IN,
                                                                                "IREF",  CP_IN,
                                                                                "VEN",   CP_IN,
                                                                                "VEP",   CP_IN,
                                                                                "VOUT",  CP_IN,
                                                                                "VG_M3", CP_IN,
                                                                                "VBIAS", CP_OUT);
// DISPLAY
CAIRO_DECLARE_PROCEDURE("DISPLAY", CP_UNCOMPLETE );

// EXPLORER
CAIRO_DECLARE_PROCEDURE("explorer", CP_COMPLETE );

// PERFORMANCES
CAIRO_DECLARE_PROCEDURE("PERFORMANCES", CP_UNCOMPLETE, "TEMP",        CP_IN,
                                                       "VDD",         CP_IN,
                                                       "VSS",         CP_IN,
                                                       "W_M1",        CP_IN,
                                                       "W_M2",        CP_IN,
                                                       "W_M3",        CP_IN,
                                                       "W_M4",        CP_IN,
                                                       "W_M5",        CP_IN,
                                                       "W_M6",        CP_IN,
                                                       "W_M7",        CP_IN,
                                                       "W_M8",        CP_IN,
                                                       "L_M1",        CP_IN,
                                                       "L_M2",        CP_IN,
                                                       "L_M3",        CP_IN,
                                                       "L_M4",        CP_IN,
                                                       "L_M5",        CP_IN,
                                                       "L_M6",        CP_IN,
                                                       "L_M7",        CP_IN,
                                                       "L_M8",        CP_IN,
                                                       "IREF",        CP_IN,
                                                       "VEN",         CP_IN,
                                                       "VEP",         CP_IN,
                                                       "VOUT",        CP_IN,
                                                       "VG_M3",       CP_IN,
                                                       "CCAP",        CP_IN,
                                                       "CL",          CP_IN,
                                                       "RL",          CP_IN,
                                                       "AD0",         CP_OUT,
                                                       "AC0",         CP_OUT,
                                                       "FT",          CP_OUT,
                                                       "PM",          CP_OUT,
                                                       "SR",          CP_OUT,
                                                       "SNE_TH",      CP_OUT,
                                                       "SNE_1/F",     CP_OUT,
                                                       "SATURATIONS", CP_OUT,
```

```
                                                        "ED0",         CP_OUT);

        // MEASURES
        CAIRO_DECLARE_PROCEDURE("MEASURES", CP_UNCOMPLETE, "TEMP",    CP_IN,
                                                           "VDD",     CP_IN,
                                                           "VSS",     CP_IN,
                                                           "W_M1",    CP_IN,
                                                           "W_M2",    CP_IN,
                                                           "W_M3",    CP_IN,
                                                           "W_M4",    CP_IN,
                                                           "W_M5",    CP_IN,
                                                           "W_M6",    CP_IN,
                                                           "W_M7",    CP_IN,
                                                           "W_M8",    CP_IN,
                                                           "L_M1",    CP_IN,
                                                           "L_M2",    CP_IN,
                                                           "L_M3",    CP_IN,
                                                           "L_M4",    CP_IN,
                                                           "L_M5",    CP_IN,
                                                           "L_M6",    CP_IN,
                                                           "L_M7",    CP_IN,
                                                           "L_M8",    CP_IN,
                                                           "IREF",    CP_IN,
                                                           "VEN",     CP_IN,
                                                           "VEP",     CP_IN,
                                                           "VOUT",    CP_IN,
                                                           "VG_M3",   CP_IN,
                                                           "CCAP",    CP_IN,
                                                           "CL",      CP_IN,
                                                           "RL",      CP_IN,
                                                           "FPND",    CP_OUT,
                                                           "FPD",     CP_OUT,
                                                           "FZ",      CP_OUT,
                                                           "POWER",   CP_OUT,
                                                           "AREA",    CP_OUT,
                                                           "VOUTMAX", CP_OUT,
                                                           "VOUTMIN", CP_OUT,
                                                           "VICMMAX", CP_OUT,
                                                           "VICMMIN", CP_OUT,
                                                           "CIN-",    CP_OUT,
                                                           "CIN+",    CP_OUT,
                                                           "COUT",    CP_OUT,
                                                           "CMRR",    CP_OUT);
CAIRO_END_CREATE(OTAS2ET)

/*************************************************************************/
/* Function: DSES                                                        */
/*************************************************************************/
/*                                                                       */
/* This is the DESIGN SPACE EXPLORATION section of the CAIRO+ generator  */
/*                                                                       */
/*************************************************************************/

CAIRO_BEGIN_DSES(OTAS2ET,char *name)

        /************************************\
        |           SIZING PROCEDURES        |
        \************************************/

        // ------------------------------------------------
        // Procedure: simulate_with_input_offset_and_feedback
        // ------------------------------------------------

          CAIRO_BEGIN_PROCEDURE("simulate_with_input_offset_and_feedback")

                double temp;
                double vdd;
                double vss;
                double l_m1;
                double l_m2;
                double l_m3;
                double l_m4;
                double l_m5;
                double l_m6;
                double l_m7;
```

```
double l_m8;
double w_m1;
double w_m2;
double w_m3;
double w_m4;
double w_m5;
double w_m6;
double w_m7;
double w_m8;
double iref;
double ven;
double vep;
double vout;
double vg_m3;

CAIRO_CREATE_EQUIPOTENTIALS;

     CAIRO_BEGIN_SYNTHESIS

        /*************************************\
        |  Reading Parameters from higher level |
        \*************************************/

        CAIRO_TRY_GET_VALUE("TEMP",temp)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("TEMP","TEMP not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M1",l_m1)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M1","L_M1 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M2",l_m2)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M2","L_M2 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M3",l_m3)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M3","L_M3 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M4",l_m4)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M4","L_M4 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M5",l_m5)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M5","L_M5 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M6",l_m6)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M6","L_M6 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M7",l_m7)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M7","L_M7 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("L_M8",l_m8)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("L_M8","L_M8 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("W_M1",w_m1)
        IF_NO_VALUE
              FATAL_ERROR_PARAM("W_M1","W_M1 not set in OTAS2ET",LOCATION) ;
        ENDIF_NO_VALUE

        CAIRO_TRY_GET_VALUE("W_M2",w_m2)
        IF_NO_VALUE
```

```
      FATAL_ERROR_PARAM("W_M2","W_M2 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M3",w_m3)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M3","W_M3 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M4",w_m4)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M4","W_M4 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M5",w_m5)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M5","W_M5 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M6",w_m6)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M6","W_M6 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M7",w_m7)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M7","W_M7 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("W_M8",w_m8)
IF_NO_VALUE
      FATAL_ERROR_PARAM("W_M8","W_M8 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("IREF", iref)
IF_NO_VALUE
      FATAL_ERROR_PARAM("IREF","IREF not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEN", ven)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VEN","VEN not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VEP", vep)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VEP","VEP not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VOUT", vout)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VOUT","VOUT not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VDD",vdd)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VDD","V not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VSS",vss)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VSS","VSS not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

CAIRO_TRY_GET_VALUE("VG_M3",vg_m3)
IF_NO_VALUE
      FATAL_ERROR_PARAM("VG_M3","VG_M3 not set in OTAS2ET",LOCATION) ;
ENDIF_NO_VALUE

// ---------------
// Synthesize TR8
// ---------------

cout << "Synthesizing TR8 ..." << endl;

CAIRO_SET_PARAM("tr_m8","TEMP",temp) ;
```

```
CAIRO_SET_PARAM("tr_m8","L",l_m8) ;
CAIRO_SET_PARAM("tr_m8","W",w_m8) ;
CAIRO_SET_PARAM("tr_m8","IDS",iref) ;
CAIRO_SET_PARAM("tr_m8","VS",vss) ;

CAIRO_COMPUTE("tr_m8","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR8") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR7
// ---------------

cout << "Synthesizing TR7 ..." << endl;

CAIRO_SET_PARAM("tr_m7","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m7","L"  ,l_m7) ;
CAIRO_SET_PARAM("tr_m7","W"  ,w_m7) ;
CAIRO_SET_PARAM("tr_m7","VD" ,vout ) ;

CAIRO_COMPUTE("tr_m7","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR7") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR6
// ---------------

cout << "Synthesizing TR6 ..." << endl;

string expr = DPARAM("tr_m6","IDS") EQ DPARAM3(-1.0,"tr_m7","IDS");
CAIRO_ADD_CONSTRAINT(expr.c_str());

CAIRO_SET_PARAM("tr_m6","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m6","L", l_m6 ) ;
CAIRO_SET_PARAM("tr_m6","W", w_m6 ) ;
CAIRO_SET_PARAM("tr_m6","VS",vdd ) ;

CAIRO_COMPUTE("tr_m6","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR6") ;
END_ON_PROCEDURE_STATUS

// --------------
// Synthesize TR4
// --------------

cout << "Synthesizing TR4 ..." << endl;

CAIRO_SET_PARAM("tr_m4","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m4","L",l_m4) ;
CAIRO_SET_PARAM("tr_m4","W",w_m4) ;
CAIRO_SET_PARAM("tr_m4","VG",vg_m3) ;

CAIRO_COMPUTE("tr_m4","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR4") ;
END_ON_PROCEDURE_STATUS

// --------------
// Synthesize TR2
// --------------

cout << "Synthesizing TR2 ..." << endl;

expr = DPARAM("tr_m2","IDS") EQ DPARAM3(-1.0,"tr_m4","IDS");
CAIRO_ADD_CONSTRAINT(expr.c_str());

CAIRO_SET_PARAM("tr_m2","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m2","L",l_m2) ;
CAIRO_SET_PARAM("tr_m2","W",w_m2) ;
CAIRO_SET_PARAM("tr_m2","VG",vep) ;
```

```
CAIRO_COMPUTE("tr_m2","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize M2") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR3
// ---------------

cout << "Synthesizing TR3 ..." << endl;

CAIRO_SET_PARAM("tr_m3","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m3","L",l_m3) ;
CAIRO_SET_PARAM("tr_m3","W",w_m3) ;

CAIRO_COMPUTE("tr_m3","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR3") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR1
// ---------------

cout << "Synthesizing TR1 ..." << endl;

expr = DPARAM("tr_m1","IDS") EQ DPARAM3(-1.0,"tr_m3","IDS");
CAIRO_ADD_CONSTRAINT(expr.c_str());

CAIRO_SET_PARAM("tr_m1","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m1","L",l_m1) ;
CAIRO_SET_PARAM("tr_m1","W",w_m1) ;
CAIRO_SET_PARAM("tr_m1","VG",ven) ;

CAIRO_COMPUTE("tr_m1","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR1") ;
END_ON_PROCEDURE_STATUS

// ---------------
// Synthesize TR5
// ---------------

cout << "Synthesizing TR5 ..." << endl;

CAIRO_SET_PARAM("tr_m5","TEMP",temp) ;
CAIRO_SET_PARAM("tr_m5","L",l_m5) ;
CAIRO_SET_PARAM("tr_m5","W",w_m5) ;

CAIRO_COMPUTE("tr_m5","synthesize");
ON_PROCEDURE_STATUS
      CAIRO_ERROR_MESSAGE("cannot synthesize TR5") ;
END_ON_PROCEDURE_STATUS

// ------------------------------------------------------------------------
// Newton Raphson Constraint: Solve Feedback VEN - F(VEN) = VEN - VOUT = 0
// ------------------------------------------------------------------------

expr = NREQ( MPARAM("VEN") WITH DPARAM("tr_m1","VG") WITH DPARAM("tr_m7","VD") );
CAIRO_ADD_CONSTRAINT(expr.c_str());

// ---------------------------------------------------------------------------------
// Newton Raphson Constraint: Solve Ids,m5(Vg,m3) - Ids,m1(Vg,m3) - Ids,m2(Vg,m3) = 0
// ---------------------------------------------------------------------------------

expr = NREQ( MPARAM("VG_M3") WITH DPARAM("tr_m5","IDS") WITH DPARAM("tr_m1","IDS") WITH DPARAM("tr_m2","IDS") ) ;
CAIRO_ADD_CONSTRAINT(expr.c_str());

// -------------------
// Synthesize device
// -------------------

CAIRO_AUTO_SIZE_AND_BIAS();

// ------------------------------------------------------------
```

```
        // Newton Raphson Constraint: Solve Vs,m1(VEP) - Vs,m2(VEP) = 0
        // -------------------------------------------------------------

        expr = NREQ( MPARAM("VEP") WITH DPARAM("tr_m1","VS") WITH DPARAM("tr_m2","VS") ) ;
        CAIRO_ADD_CONSTRAINT(expr.c_str());

        // -------------------
        // Display Graphs
        // -------------------

        CAIRO_DISPLAY_GRAPHS();

CAIRO_END_SYNTHESIS

// -------------------
// Execute design plan
// -------------------

CAIRO_EXECUTE_DESIGN_PLAN_USING_NR(true);

// -------------------
// Display Rules
// -------------------

CAIRO_DISPLAY_RULES();

// ----------------------
// Set Number of fingers
// ----------------------

CAIRO_SET_PARAM("tr_m1","M_VALUE",4L);

CAIRO_SET_PARAM("tr_m2","M_VALUE",4L);

CAIRO_SET_PARAM("tr_m3","M_VALUE",2L);

CAIRO_SET_PARAM("tr_m4","M_VALUE",2L);

CAIRO_SET_PARAM("tr_m6","M_VALUE",20L);

CAIRO_SET_PARAM("tr_m7","M_VALUE",16L);

CAIRO_SET_PARAM("tr_m5","M_VALUE",4L);

CAIRO_SET_PARAM("tr_m8","M_VALUE",4L);


// ----------------------------
// Call direct size for devices
// ----------------------------

CAIRO_COMPUTE("tr_m1","direct_size");

CAIRO_COMPUTE("tr_m2","direct_size");

CAIRO_COMPUTE("tr_m3","direct_size");

CAIRO_COMPUTE("tr_m4","direct_size");

CAIRO_COMPUTE("tr_m6","direct_size");

CAIRO_COMPUTE("tr_m7","direct_size");

CAIRO_COMPUTE("tr_m5","direct_size");

CAIRO_COMPUTE("tr_m8","direct_size");

// ------------------------
// Return polarization
// ------------------------

double vbias = 0.0;

CAIRO_GET_PARAM("tr_m5","VG",vbias);
```

```
        CAIRO_SET_VALUE("VBIAS", vbias, CP_VALID) ;

        CAIRO_GET_VALUE("VG_M3", vg_m3) ;

        cout << "VG_M3 value is " << vg_m3 << endl;

        CAIRO_GET_VALUE("VEN", ven) ;

        cout << "VEN value is " << ven << endl;

        CAIRO_GET_VALUE("VEP", vep) ;

        cout << "VEP value is " << vep << endl;

        CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ----------------------------
// Procedure: explorer
// ----------------------------

CAIRO_BEGIN_PROCEDURE("explorer")

            CAIRO_BEGIN_EXPLORE

      // -------------------
      // Execute IE
      // -------------------

      CAIRO_IE_USE();

      CAIRO_IE_ADD_INDEPENDENT_PARAMETERS();

      // Performances
      CAIRO_IE_ADD_WATCH("PERFORMANCES","AD0");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","AC0");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","SATURATIONS");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","ED0");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","SNE_TH");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","SNE_1/F");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","PM");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","SR");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","FT");
      CAIRO_IE_ADD_WATCH("PERFORMANCES","KCL_BIAS");

      // Measures
      CAIRO_IE_ADD_WATCH("MEASURES","CMRR");
      CAIRO_IE_ADD_WATCH("MEASURES","FPND");
      CAIRO_IE_ADD_WATCH("MEASURES","FPD");
      CAIRO_IE_ADD_WATCH("MEASURES","FZ");
      CAIRO_IE_ADD_WATCH("MEASURES","CIN-");
      CAIRO_IE_ADD_WATCH("MEASURES","CIN+");
      CAIRO_IE_ADD_WATCH("MEASURES","COUT");
      CAIRO_IE_ADD_WATCH("MEASURES","VICMMIN");
      CAIRO_IE_ADD_WATCH("MEASURES","VICMMAX");
      CAIRO_IE_ADD_WATCH("MEASURES","VOUTMIN");
      CAIRO_IE_ADD_WATCH("MEASURES","VOUTMAX");
      CAIRO_IE_ADD_WATCH("MEASURES","POWER");
      CAIRO_IE_ADD_WATCH("MEASURES","AREA");

      // Wrappers
      CAIRO_IE_SET_DISPLAY_WRAPPER("DISPLAY");
      CAIRO_IE_SET_DISPLAY_FUNCTION("ALL(VGS,W,L)");

      CAIRO_IE_DISPLAY(name);

      CAIRO_IE_RELEASE();

      CAIRO_END_EXPLORE

      if (CAIRO_IE_EXIT == false)
      {
            // ---------------------
            // Set Number of fingers
```

```
            // ----------------------

            CAIRO_SET_PARAM("tr_m1","M_VALUE",4L);

            CAIRO_SET_PARAM("tr_m2","M_VALUE",4L);

            CAIRO_SET_PARAM("tr_m3","M_VALUE",2L);

            CAIRO_SET_PARAM("tr_m4","M_VALUE",2L);

            CAIRO_SET_PARAM("tr_m6","M_VALUE",20L);

            CAIRO_SET_PARAM("tr_m7","M_VALUE",16L);

            CAIRO_SET_PARAM("tr_m5","M_VALUE",4L);

            CAIRO_SET_PARAM("tr_m8","M_VALUE",4L);

            // ------------------
            // Execute design plan
            // ------------------

            CAIRO_EXECUTE_DESIGN_PLAN_USING_NR(true);

            // ----------------------------
            // Call direct size for devices
            // ----------------------------

            CAIRO_COMPUTE("tr_m1","direct_size");

            CAIRO_COMPUTE("tr_m2","direct_size");

            CAIRO_COMPUTE("tr_m3","direct_size");

            CAIRO_COMPUTE("tr_m4","direct_size");

            CAIRO_COMPUTE("tr_m6","direct_size");

            CAIRO_COMPUTE("tr_m7","direct_size");

            CAIRO_COMPUTE("tr_m5","direct_size");

            CAIRO_COMPUTE("tr_m8","direct_size");

            // ------------------------
            // Return polarization
            // ------------------------

            double vep = 0.0;
            double ven = 0.0;

            CAIRO_GET_PARAM("tr_m1","VG",ven);
            CAIRO_GET_PARAM("tr_m2","VG",vep);

            cout << " VEP = " << vep << endl;
            cout << " VEN = " << ven << endl;
      }

      CAIRO_SET_RETURN_PROCEDURE(CP_OK);

END_PROCEDURE

// ------------------------------
// Procedure: DISPLAY
// ------------------------------

CAIRO_BEGIN_PROCEDURE("DISPLAY")

      // ------------------------------------
      // Calculate small signales for TR1
      // ------------------------------------

      double vs_m1;
      double vd_m1;
      double vg_m1;
```

```
            double vb_m1;

            CAIRO_GET_PARAM("tr_m1","VD",vd_m1);
            CAIRO_GET_PARAM("tr_m1","VS",vs_m1);
            CAIRO_GET_PARAM("tr_m1","VG",vg_m1);
            CAIRO_GET_PARAM("tr_m1","VB",vb_m1);

            CAIRO_SET_PARAM("tr_m1","VDS", vd_m1 - vs_m1);
            CAIRO_SET_PARAM("tr_m1","VBS", vb_m1 - vs_m1);
            CAIRO_SET_PARAM("tr_m1","VGS", vg_m1 - vs_m1);

            CAIRO_SET_PARAM("tr_m1","SIF_FREQ", 1.0);

            CAIRO_COMPUTE("tr_m1","ALL(VGS,W,L)");

            . . .

            // -----------------------------------
            // Return code
            // -----------------------------------

            CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

        // ------------------------------
        // Procedure: PERFORMANCES
        // ------------------------------

        CAIRO_BEGIN_PROCEDURE("PERFORMANCES")

             // Compute primary performances here
             . . .

             CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

        // ------------------------------
        // Procedure: MEASURES
        // ------------------------------

        CAIRO_BEGIN_PROCEDURE("MEASURES")

             // Compute secondary performances here
             . . .

             CAIRO_SET_RETURN_PROCEDURE(CP_OK);

        END_PROCEDURE

        // ---------------------------
        // ------------- Default
        // ---------------------------

        CAIRO_DEFAULT_PROCEDURE
              FATAL_ERROR_PROCEDURE(PROCEDURE_NAME,"unknown procedure",LOCATION) ;
        END_DEFAULT_PROCEDURE

CAIRO_END_DSES(OTAS2ET)

/*************************************************************************/
/* Function: LAYOUT                                                      */
/*************************************************************************/
/*                                                                       */
/* This is the ROUTE section of the CAIRO+ generator                     */
/*                                                                       */
/*************************************************************************/

CAIRO_BEGIN_LAYOUT(OTAS2ET, char *name)

     // Perform procedural routing here
     . . .

CAIRO_END_LAYOUT(OTAS2ET)
```

Appendix H

Knowledge-Aware Synthesis Code for the OTA Amplifier

```
/************************************************************************/
/*                     File : test_OPTIMIZE_OTA.cpp                     */
/*                                                                      */
/* Description : File for optimizing the OTA amplifier                  */
/*                                                                      */
/* Language    : C++               Version : 1.0                        */
/*                                                                      */
/* Author      : Ramy ISKANDER                                          */
/*                                                                      */
/* License     : QPL                                                    */
/*                                                                      */
/* History     :                                                        */
/*                                                                      */
/* Function    : Knowledge-Aware synthesis of the OTA amplifier         */
/*                                                                      */
/************************************************************************/

#include "cairoplus.h"

int main(int argc, char *argv[])
{
        // ---------------- //
        // Create new chip  //
        // ---------------- //

        CAIRO_NEW_CHIP("test_OPTIMIZE_OTA") ;

        // ------------------------------------------------------------//
        // Create an amplifier instance "OTA2ET" from model "OTAS2ET"  //
        // in module library "libOTA2ET_iref"                          //
        // ------------------------------------------------------------//

        CAIRO_CREATE("libOTA2ET_iref","OTAS2ET","OTA2ET",TRANSN, true) ;

        // --------------------------------- //
        // Set initial values for parameters  //
        // --------------------------------- //

        CAIRO_SET_PARAM("OTA2ET","TEMP", 300.15) ;
        CAIRO_SET_PARAM("OTA2ET","VDD",        1.2) ;
        CAIRO_SET_PARAM("OTA2ET","VSS",        0.0) ;
        CAIRO_SET_PARAM("OTA2ET","L_BIAS", 2.0e-6) ;
        CAIRO_SET_PARAM("OTA2ET","L_DP", 2.0e-6) ;
        CAIRO_SET_PARAM("OTA2ET","L_CM", 2.0e-6) ;
        CAIRO_SET_PARAM("OTA2ET","L_M6", 2.0e-6) ;
        CAIRO_SET_PARAM("OTA2ET","VSP",        0.6) ;
        CAIRO_SET_PARAM("OTA2ET","VEG_BIAS", 0.12) ;
        CAIRO_SET_PARAM("OTA2ET","VEG_DP", 0.12) ;
```

```
CAIRO_SET_PARAM("OTA2ET","VEG_CM", -0.12) ;
CAIRO_SET_PARAM("OTA2ET","VEG_M6", -0.1) ;
CAIRO_SET_PARAM("OTA2ET","VMC", 0.6) ;
CAIRO_SET_PARAM("OTA2ET","K", 3.0) ;
CAIRO_SET_PARAM("OTA2ET","CL", 3.0e-12) ;
CAIRO_SET_PARAM("OTA2ET","RL", 1.0e6 ) ;
CAIRO_SET_PARAM("OTA2ET","CCAP", 3.0e-12) ;
CAIRO_SET_PARAM("OTA2ET","IBIAS", 25.0e-6) ;

// ------------------------------------------------- //
// Call the synthesis procedure to synthesize with offset //
// ------------------------------------------------- //

CAIRO_COMPUTE("OTA2ET","synthesize_with_offset");

// ------------------------------------------------- //
// Setup domain of the parameters for optimization   //
// ------------------------------------------------- //

CAIRO_SET_PARAM_DOMAIN("OTA2ET","L_BIAS", 0.2e-6, 3e-6, 0.1e-6) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","L_DP", 0.2e-6, 3e-6, 0.1e-6) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","L_CM", 0.2e-6, 3e-6, 0.1e-6) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","L_M6", 0.2e-6, 3e-6, 0.1e-6) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","VEG_BIAS", 0.01, 0.2, 0.01) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","VEG_DP", 0.01, 0.2, 0.01) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","VEG_CM", -0.2, -0.01, 0.01) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","VEG_M6", -0.2, -0.01, 0.01) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","K", 1.0,5.0,0.5) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","CCAP", 1.0e-12, 5.0e-12, 0.1e-12) ;
CAIRO_SET_PARAM_DOMAIN("OTA2ET","IBIAS", 10.0e-6, 30.0e-6, 1.0e-6) ;

// -------------------------------- //
// Setup constraints on parameters   //
// -------------------------------- //

CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","AD0", CP_ABOVE, 65.0) ;        // in dB
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","AC0", CP_BELOW, 17.0) ;        // in dB
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","FT", CP_ABOVE, 6.0e6) ;        // in MHz
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","PM", CP_ABOVE, 76.0) ;         // in degree
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","SNE_1HZ", CP_BELOW, 20.0e-6) ; // in V/SQRT(Hz)
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","SNE_FT", CP_BELOW, 2.0e-8) ;   // in V/SQRT(Hz)
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","SR", CP_ABOVE, 6.0e6) ;        // in V/?S
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","SATURATIONS", CP_EQUAL, 8.0) ; // Unitless
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","POWER", CP_BELOW, 0.001 ) ;    // in Watts
CAIRO_SET_PARAM_CONSTRAINT("OTA2ET","ED0", CP_BELOW, 0.002 ) ;      // in Watts

// ----------------- //
// Start Optimizer   //
// ----------------- //

CAIRO_OPTIMIZER_USE("OTA2ET");

// ----------------------------- //
// Setup procedures to optimize  //
// ----------------------------- //

CAIRO_OPTIMIZE_PROCEDURE("OTA2ET","PERFORMANCES");

// ----------------- //
// Create CPU Timer  //
// ----------------- //

time_t start_time;
time_t end_time;

start_time = time(0);

// ------------------- //
// Optimize in a loop  //
// ------------------- //

CAIRO_OPTIMIZE("OTA2ET", 100, CP_SYNTHESIS);
```

```
        // ---------------- //
        // End clock timer  //
        // ---------------- //

        end_time = time(0);

        // ------------------ //
        // Display CPU Time   //
        // ------------------ //

        double cpu_time_used = difftime(end_time,start_time) ;

        cout << "-------------------------------" << endl;
        cout << "OPTIMIZATION CPU TIME: " <<  cpu_time_used << " secs" << endl;
        cout << "-------------------------------" << endl;

        // ------------------ //
        // Release Optimizer  //
        // ------------------ //

        CAIRO_OPTIMIZER_RELEASE("OTA2ET");

        // ----------------------- //
        // Set priority procedure  //
        // ----------------------- //

        CAIRO_SET_PRIORITY_PROCEDURE("OTA2ET","explorer");

        // --------------- //
        // Reshape layout  //
        // --------------- //

        CAIRO_RESHAPE() ;

        // ---------------------------------- //
        //  Display Influence Exploration Tool //
        // ---------------------------------- //

        while (CAIRO_IE_EXIT == false)
        {
            CAIRO_RESET_STABILITY();

            CAIRO_COMPUTE("OTA2ET","explorer");

            if (CAIRO_IE_EXIT == false)
            {
                if (CAIRO_IE_STABILITY)
                {
                     CAIRO_STABILITY();
                }
            }
        }

        // --------------- //
        // Save new chip  //
        // --------------- //

        CAIRO_SAVE_CHIP("test_OPTIMIZE_OTA") ;

        return(0) ;
}
```

Appendix I

Graphical User Interfaces for Modules

I.1 Influence Exploration Tool

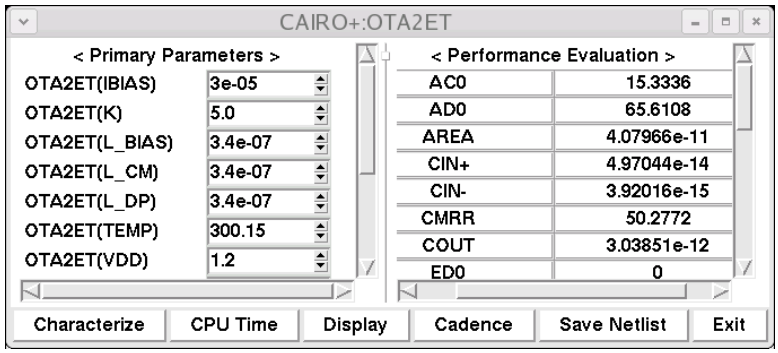

Figure I.1: *Influence Explorer User Interface*.

One potential application driven by the generation of module dependency graphs is the automatic creation of a simple influence exploration tool that can characterize circuit performances given the input parameters of the module dependency graph. The input parameters are the minimum number of parameters that has to be specified in order to fully evaluate the module dependency graph. The input parameters are the set of graph nodes that has no incident arcs. This set appears in the left pane of the interface window shown in Fig. I.1. The right pane shows all the performances evaluated using values from the left pane. Some buttons exist, namely <Characterize>, <CPU Time>, <Display>, <Cadence>, <Save Netlist> and <Exit>:

- <**Characterize**> **button:** evaluates the module dependency graphs for the values of the input parameters given in the left pane. The graph can be evaluated in either the designer mode or the simulator mode. It can generate the layout of the module after completing the graph evaluation.
- <**CPU Time**> **button:** displays the time taken to evaluate the whole graph.

- <**Display**> **button:** displays all DC and small signal parameters for the reference transistors in all instantiated devices of the module.
- <**Cadence**> **button:** back-annotates the computed dimensions in the schematic view of the module in Cadence.
- <**Save Netlist**> **button:** saves the sized hierarchical netlist of the module in spice format for the used technology.
- <**Exit**> **button:** exits the interface.

An example code of an OTA amplifier for starting an influence exploration tool will be briefly illustrated in Fig. I.3:

1. In line 1, the definition of the procedure called *EXPLORER* is started
2. In line 3, the start of an exploration section is marked.
3. In line 5, an influence explorer is allocated for use.
4. In line 7, all the input parameters with no incident arcs are added to the left pane of the influence exploration tool.
5. In lines 9-17, all primary performances are asked to be computed using the procedure *PERFORMANCES* and added to the right pane of the influence exploration tool.
6. In lines 19-25, all secondary measurements are asked to be computed using the procedure *MEASURES* and added to the right pane of the influence exploration tool.
7. In line 28, the name of a procedure is supplied to compute the small signal parameters. Here, it has the name *DISPLAY*.
8. In line 29, display all the input and output parameters bound to the predefined CAIRO+ procedure *ALL(VGS,W,L)*.
9. In line 30, display the main window of the influence exploration tool.
10. In line 32, release all resources used by the influence exploration tool.
11. In line 34, mark the end of an exploration section.
12. In line 36-40, if the <Exit> button was not pressed, the module dependency graph is re-evaluated.
13. In line 42, the procedure returns a success code to the caller.
14. In line 44, the procedure definition ends.

I.2 Displaying Graphs Using GOBLIN

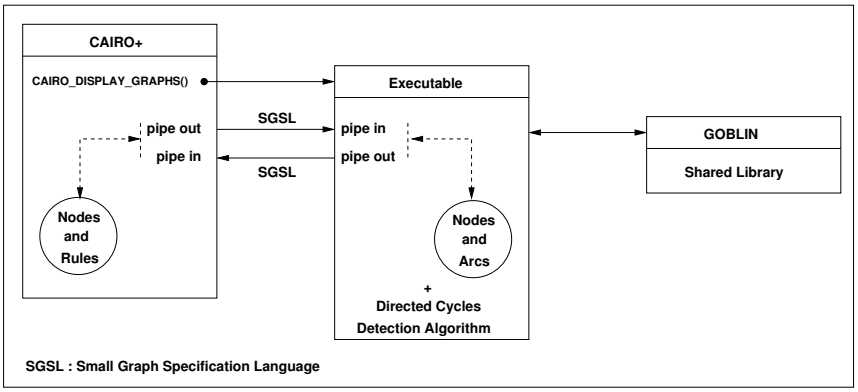

Figure I.2: *Architectural independence from graphical packages*.

Numerous graph packages that deal with all of the standard graph optimization problem exist in LGPL license. After studying many packages, we chose the GOBLIN project [Fremuth-Paeger]. The GOBLIN project consists of a C++ class library for a large series of graph optimization problems, called *GOSH*, an extension of the Tcl/Tk scripting language to graph objects, and *GOBLET*, a graphical user interface to the library functions. GOBLET includes a graph editor and supports the standard graph layout methods.

One potential problem is how to make CAIRO+ framework independent from the GOBLIN architecture and internal data structures. To achieve this, an executable program is developed for each independent graph package. This executable deals with the internal aspects of the package like data structures, package API, algorithms, drawing, ... etc. In Fig. I.2, the executable links to the shared library of GOBLIN and deals with GOBLIN graph manipulation details inside the executable. The executable is expected to have two standard bidirectional pipes. The first pipe send commands issued from CAIRO+ to the executable. The commands are issued using simple graph commands that were defined for this purpose, called *Small Graph Specification Language (SGSL)*. The executable reads each command from its input pipe and executes a special routine related to the command. The output of the command execution is sent to CAIRO+ through the second pipe. Again, SGSL is used to specify the input to CAIRO+. The mechanism implements a handshaking protocol between CAIRO+ and the executable. It is important to note that the graph representation in CAIRO+ is at higher level of abstraction. It deals with dependency nodes and dependency rules which are not supported by GOBLIN. The mechanism allows to map the nodes and rules of the dependency graphs in CAIRO to the nodes and arcs in GOBLIN graphs.

As an example for implanting algorithms is shown in Fig. I.2, the direct cycles detection algorithm [Tiernan70] was developed in the executable which deals with simple nodes and arcs. It is called during dependency graph display. The detected directed cycles are sent back to CAIRO+ and mapped to the original dependency graph.

```
1   CAIRO_BEGIN_PROCEDURE(”explorer”)
2
3       CAIRO_BEGIN_EXPLORE
4
5           CAIRO_IE_USE();
6
7           CAIRO_IE_ADD_INDEPENDENT_PARAMETERS();
8
9           // Performances
10          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”AD0”);
11          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”AC0”);
12          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”ED0”);
13          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”SNE_1HZ”);
14          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”SNE_FT”);
15          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”PM”);
16          CAIRO_IE_ADD_WATCH(”PERFORMANCES”,”SR”);
17          . . .
18
19          // Measures
20          CAIRO_IE_ADD_WATCH(”MEASURES”,”CMRR”);
21          CAIRO_IE_ADD_WATCH(”MEASURES”,”VICMMIN”);
22          CAIRO_IE_ADD_WATCH(”MEASURES”,”VICMMAX”);
23          CAIRO_IE_ADD_WATCH(”MEASURES”,”VOUTMIN”);
24          CAIRO_IE_ADD_WATCH(”MEASURES”,”VOUTMAX”);
25          . . .
26
27          // Display IE
28          CAIRO_IE_SET_DISPLAY_WRAPPER(”DISPLAY”);
29          CAIRO_IE_SET_DISPLAY_FUNCTION(”ALL(VGS,W,L)”);
30          CAIRO_IE_DISPLAY(name);
31
32          CAIRO_IE_RELEASE();
33
34      CAIRO_END_EXPLORE
35
36      if (CAIRO_IE_EXIT == false)
37      {
38          CAIRO_EXECUTE_DESIGN_PLAN();
39          . . .
40      }
41
42      CAIRO_SET_RETURN_PROCEDURE(CP_OK);
43
44  END_PROCEDURE
```

Figure I.3: *Example code for using an influence exploration tool.*

Appendix J

Module Dependency Graphs of the Fully Differential Transconductor

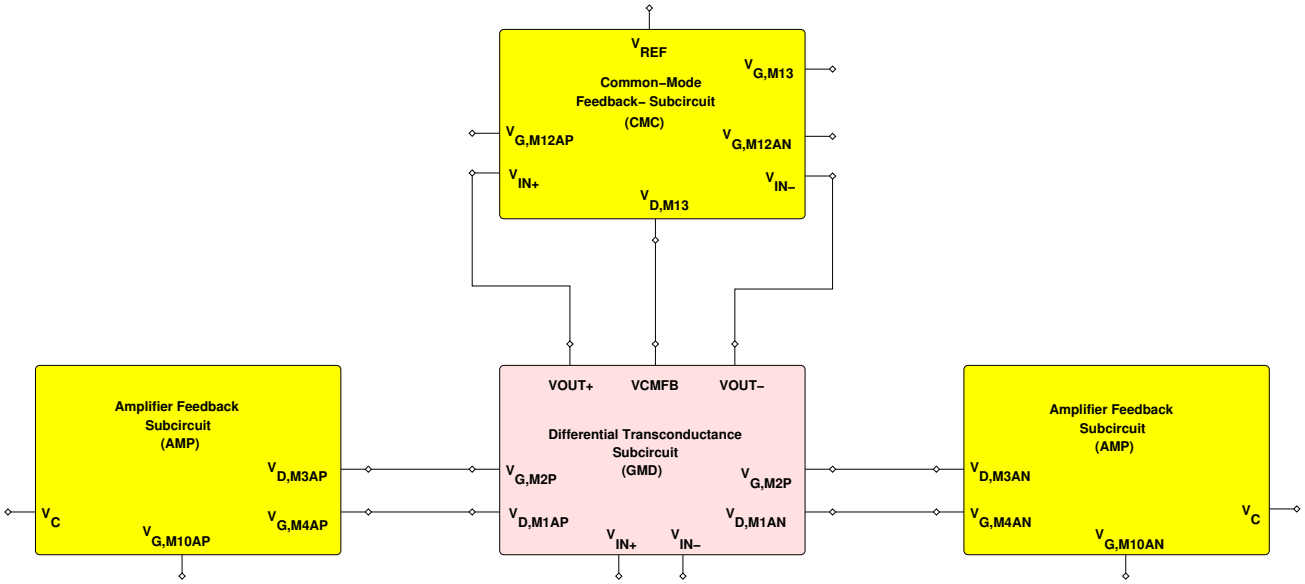

Figure J.1: *Hierarchy of subcircuits for the transconductor.*

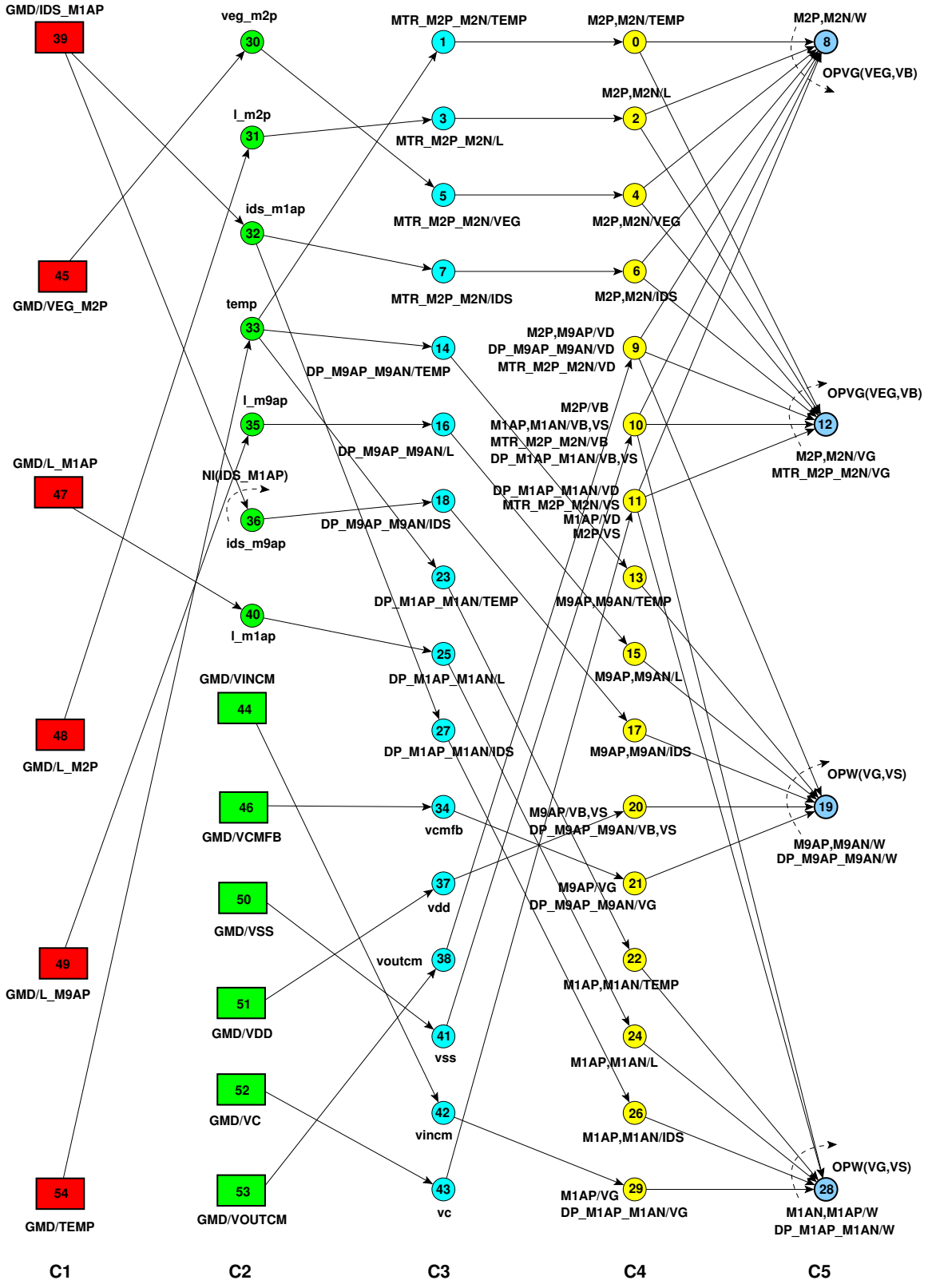

Figure J.2: *Module dependency graph of GMD of the fully differential transconductor in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

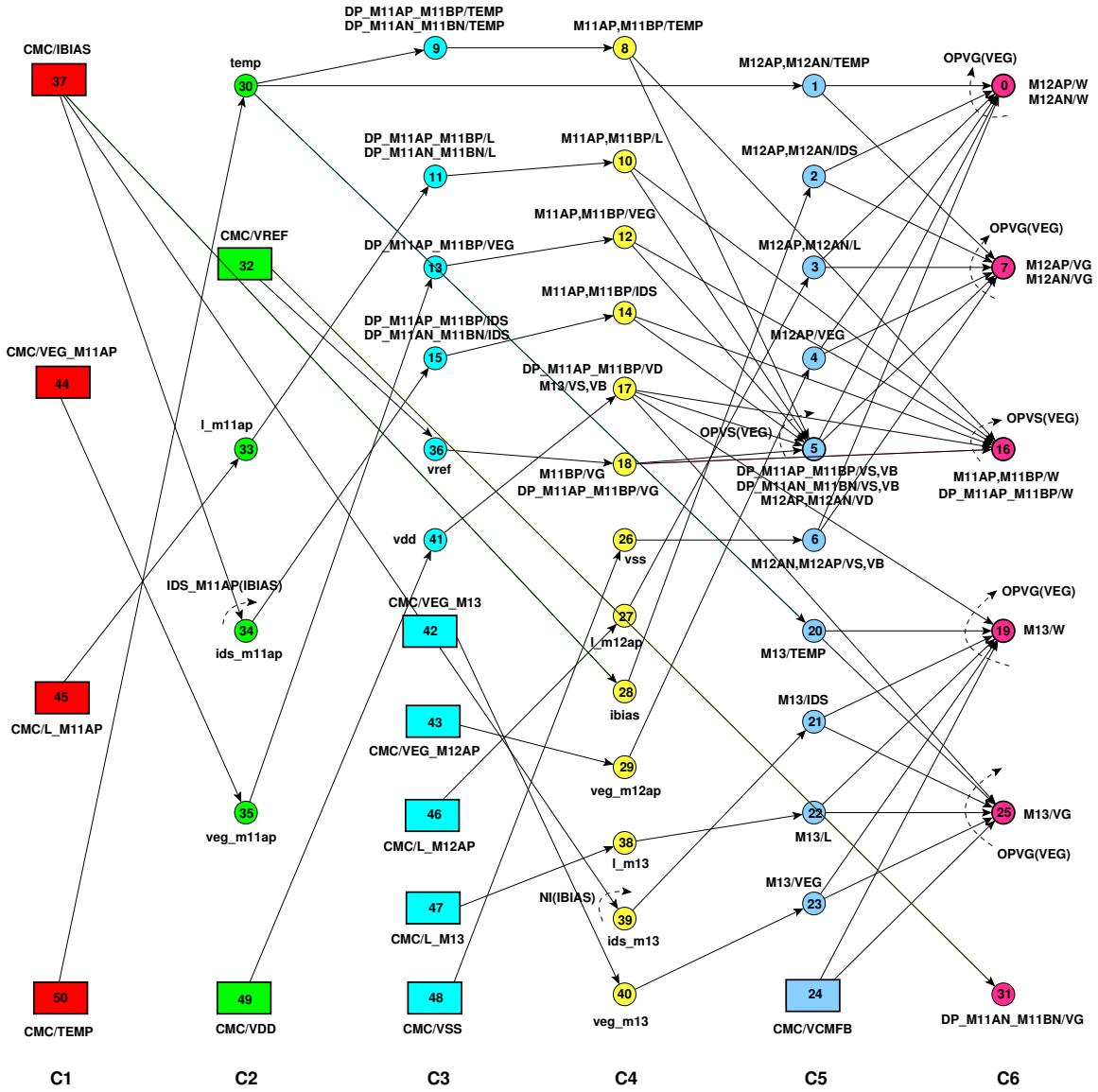

Figure J.3: *Module dependency graph of CMC of the fully differential transconductor in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

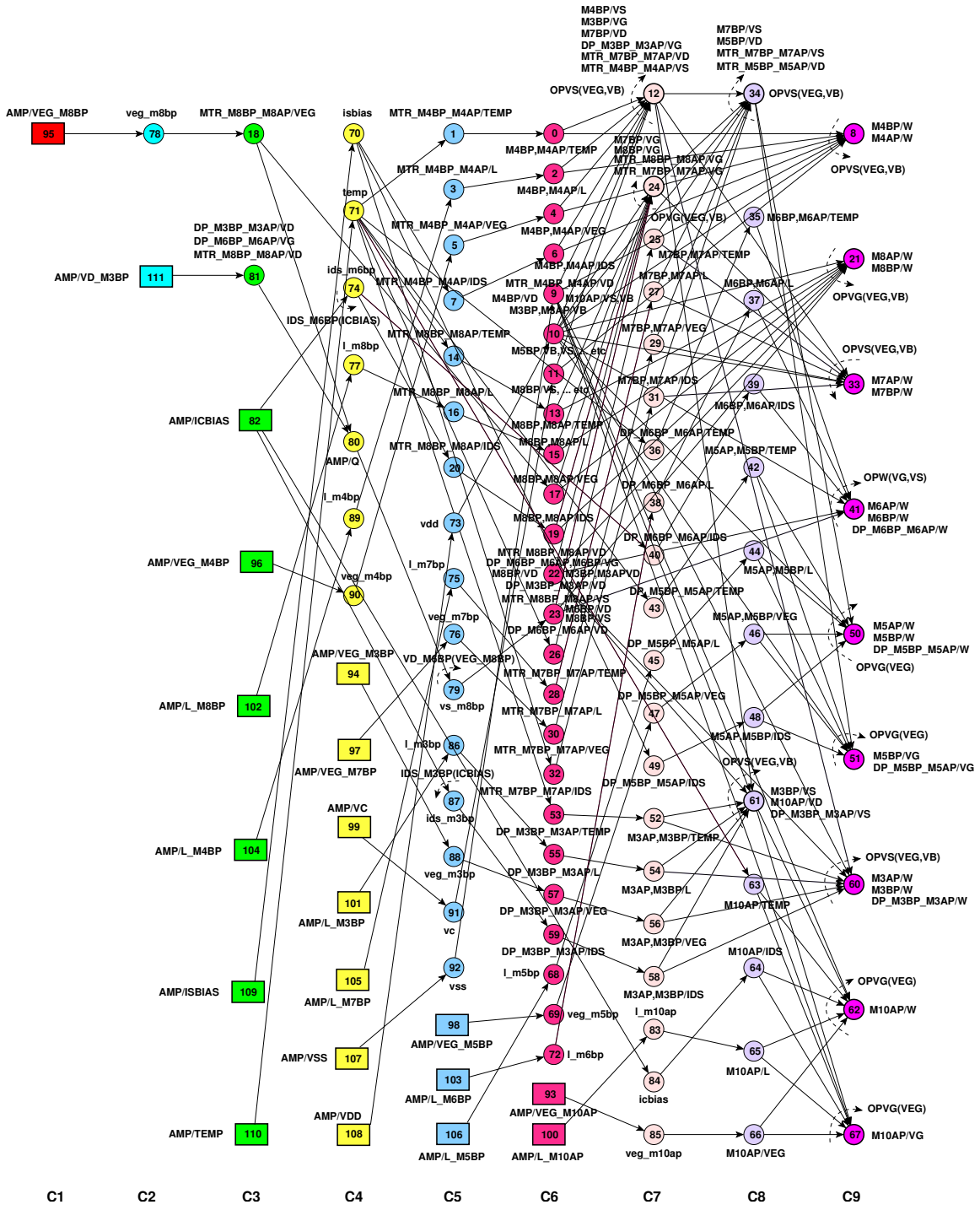

Figure J.4: *Module dependency graph of AMP of the fully differential transconductor in designer mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

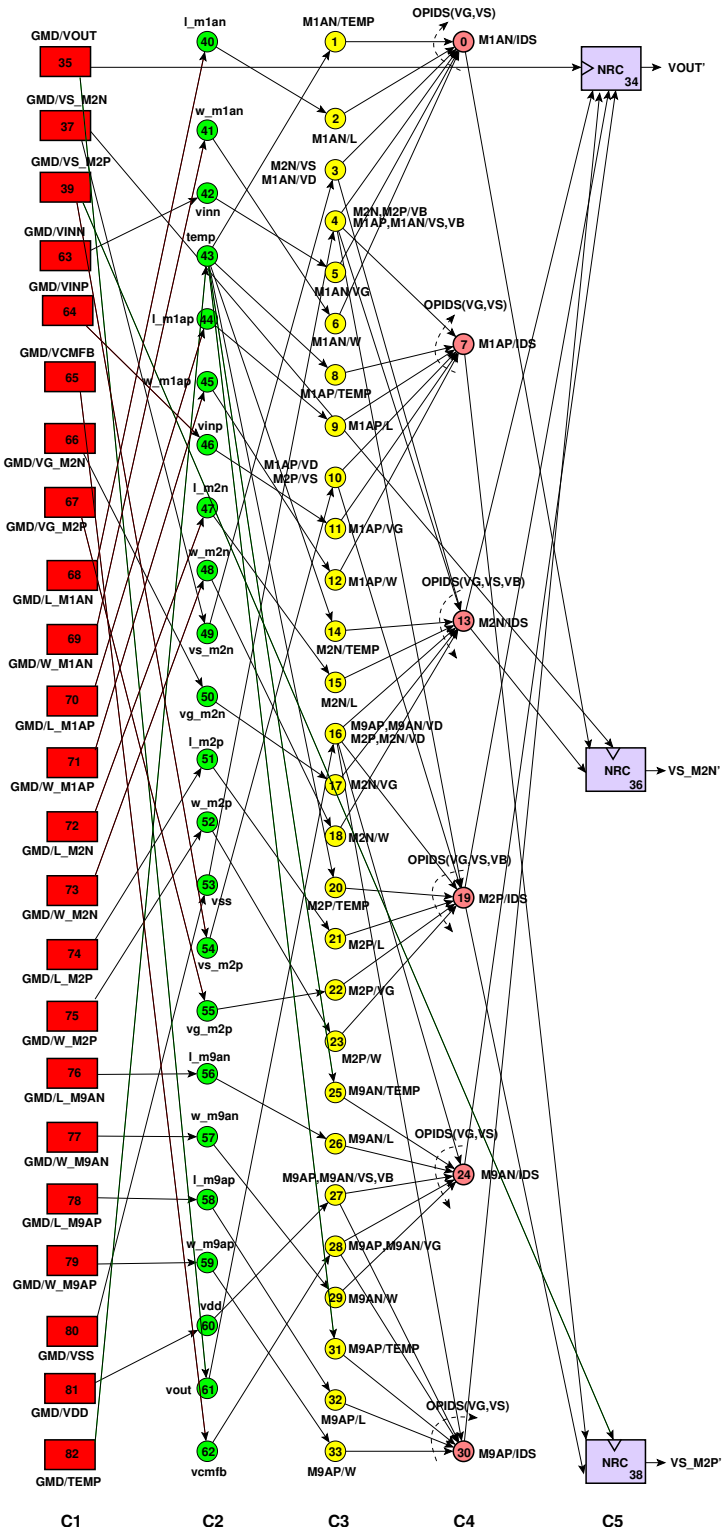

Figure J.5: *Module dependency graph of GMD of the fully differential transconductor in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

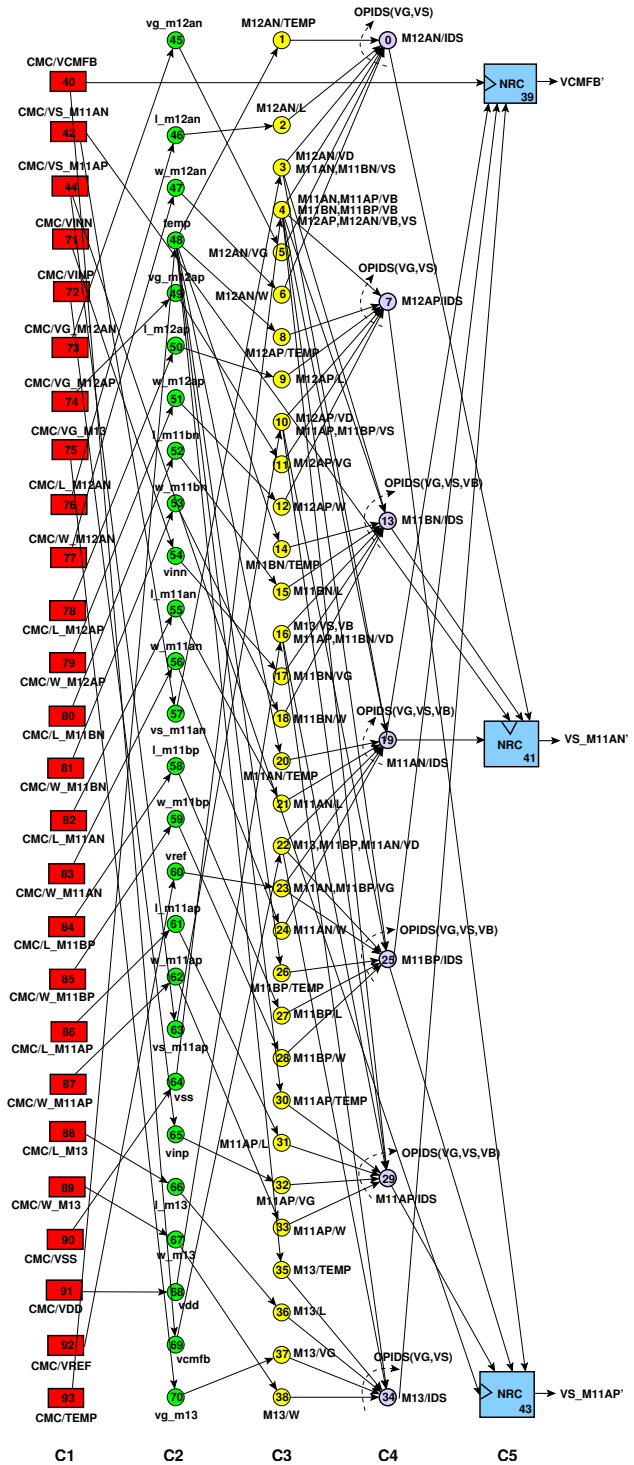

Figure J.6: *Module dependency graph of CMC of the fully differential transconductor in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

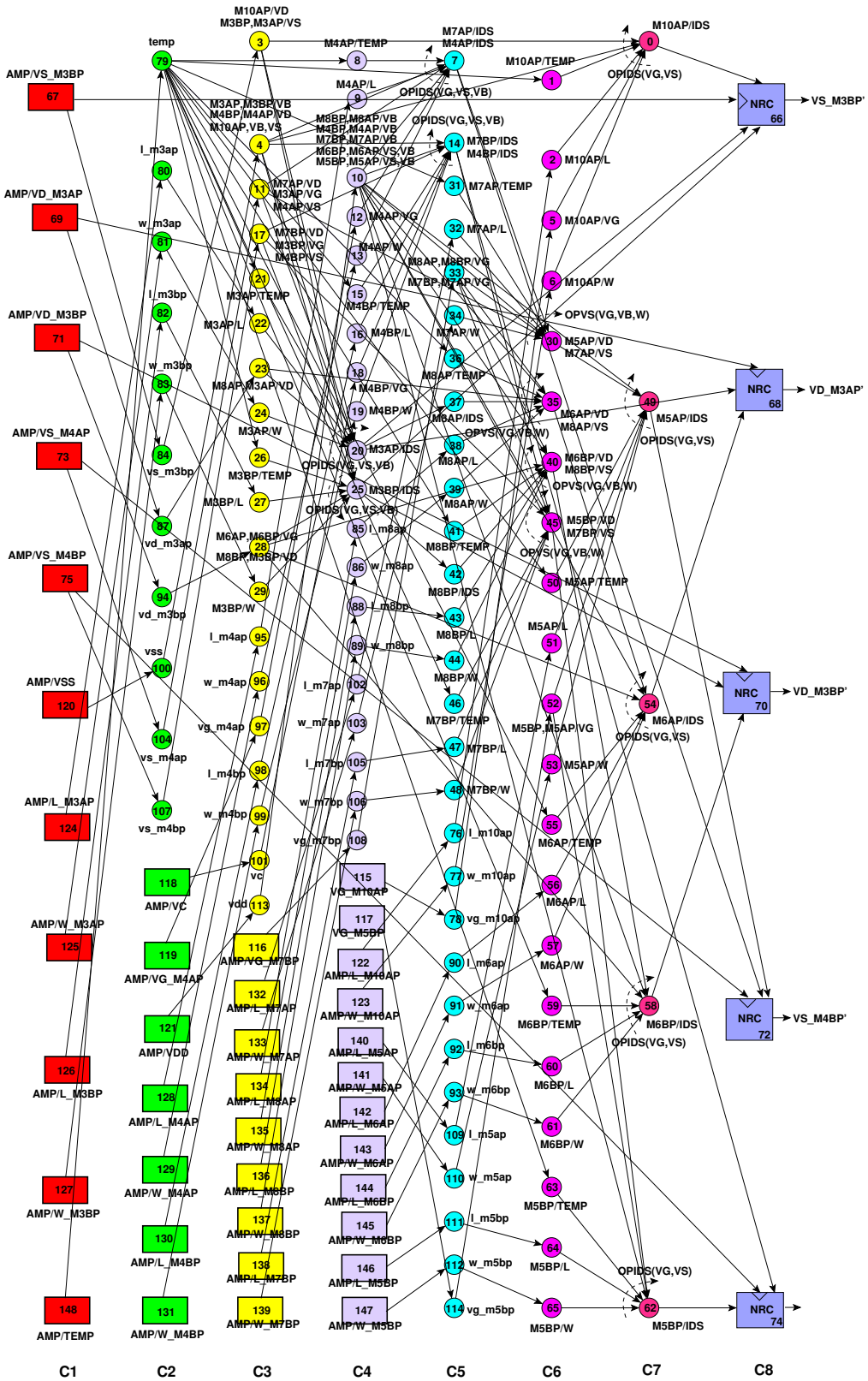

Figure J.7: *Module dependency graph of AMP of the fully differential transconductor in simulator mode: (a) Rectangles are amplifier parameters, (b) Thin circles are variables and parameters used for parameter mapping, (c) bold circles with arrows are operators, (d) Thin circles with arrows are DDPs. Each node is a represented by a triplet (column, name, index). Device connectors, equipotentials and weights are not shown for clarity.*

List of Publications

The following publications can be downloaded from: http://www-asim.lip6.fr/publications/.

Methodology and CAD Tools

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Automatic DC Operating Point Computation and Design Plan Generation for Analog IPs", *Analog Integrated Circuits and Signal Processing Journal.*, Vol. 56, No. 1-2, pp. 93-105, August 2008.

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Computing Systematic Offset in Amplifiers Using Hierarchical Graph-Based Sizing and Biasing", The 19th IEEE International Conference on MicroElectronics (ICM'07), pp. Cairo, Egypt, December 2007.

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Détection et évaluation des tensions de décalage d'un circuit analogique", *8ème Colloque sur le Traitement Analogique de l'Information, du Signal et ses Applications (TAISA'07)*, Lyon, France, Octobre 2007.

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Systematic Offset Detection and Evaluation Using Hierarchical Graph-Based Sizing and Biasing", *The 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS'07)*, Marrakech, Morocco, December 2007,.

- Ramy Iskander, Dimitri Galayko, Marie-Minerve Louërat and Andreas Kaiser, "Knowledge-Aware Synthesis of Analog Cells using Hierarchical Graph-Based Sizing and Biasing, *The 50th IEEE MidWest Symposium on Circuits and Systems (MWSCAS07)*, Québéc, Canada, August 2007.

- Ramy Iskander, Dimitri Galayko, Marie-Minerve Louërat et Andreas Kaiser, "Connaissance et Optimisation pour la Synthése Analogique", *1er Colloque National du GDR SOC-SIP*, Paris, France, Juin 2007.

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Hierarchical Graph-Based Sizing for Analog Cells Through Reference", *IEEE Ph.D. Research in Microelectronics and Electronics (PRIME'06)*, Otronto, Italy, June 2006. Winner of the Bronze Leaf Certificate.

- Ramy Iskander, Marie-Minerve Louërat et Andreas kaiser, "Dimensionnement automatique d'un circuit analogique à l'aide des transistors de référence", *7ème Colloque sur le Traitement Analogique de l'Information, du Signal et ses Applications (TAISA'06)*, Strasbourg, France, Octobre 2006.

- Ramy Iskander, Marie-Minerve Louërat and Andreas Kaiser, "Automatic Biasing Point Extraction and Design Plan Generation for Analog IPs", *48th IEEE MidWest Symposium on Circuits and Systems (MWSCAS'05)*, Cincinnati, Ohio USA, August 2005. Selected for Publication in Analog Integrated Circuits and Signal Processing Journal.

- Ramy Iskander, Laurent de Lamarre, Pierre Nguyen Tuong, Andreas Kaiser et Marie-Minerve Louërat, "Synthése d'un IP Amplificateur Analogique CMOS avec CAIRO+", *6éme Colloque sur le Traitement Analogique de l'Information, du Signal et ses Applications (TAISA'05)*, Marseille, France, Octobre 2005.

- Ramy Iskander, Laurent de Lamarre, Andreas Kaiser and Marie-Minerve Louërat, "Design Space Exploration for Analog IPs using CAIRO+", *The International Conference on Electrical Electronic and Computer Engineering 2004 (ICEEC'04)*, pp. 473-476, Cairo, Egypt, September 2004.

Additional Publications

- Dimitri Galayko, Ramy Iskander, Marie-Minerve Louërat et Alain Greiner, "Réutilisation et migration d'amplificateurs avec CAIRO+", *Coordination Nationale pour la Formation en Micro-nanoélectronique (CNFM'06)*, Saint Malo, France, Novembre 2006.

- Ramy Iskander, Mohamed Dessouky, Maie Aly, Mahmoud Magdy, Noha Hassan, Noha Soliman, and Sami Moussa, "Synthesis of CMOS Analog Cells Using AMIGO", *Design Automation and Test in Europe (DATE'03)*, vol. 02, no. 2, pp. 20297-20302, Messe Munich, Germany, March 2003.

- Aida El-Sabban, Ramy Iskander, Hisham Haddara, and Hani Ragai, "GA-Based Analog Synthesis of CMOS Power Amplifiers in the 2.45 GHz Band", *Mediterranean Microwave Symposium (MMS'03)*, Cairo, Egypt, May 2003.

- Ramy Iskander, Somaya Kayed and Hany Ragai, "Study of Technology Migration on 2nd Generation Current Conveyors Performance, *IEEE 44th MidWest Symposium On Circuits And Systems (MWSCAS'01)*, Vol. 01, pp. 286-289, Dayton, Ohio, August 2001.

Bibliography

[Aboushady01] H. Aboushady and M.-M. Louërat. Low-power design of low-voltage current-mode integrators for continuous-time $\Delta\Sigma$ modulators. *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 276–279, May 2001.

[Acosta02] R. Acosta, F. Silveira, and P. Aguirre. Experiences on analog circuit technology migration and reuse. *Proc. of the* 15^{th} *Symposium on integrated circuits and Systems Design*, pages 169–174, September 2002.

[Balarin03] F. Balarin, Y. Watanabe, and *et al*. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, April 2003.

[Banu88] M. Banu, J. M. Khoury, and Yannis Tsividis. Fully differential operational amplifiers with accurate output balancing. *IEEE J. of Solid-State Circuits*, 23(6):1410–1414, December 1988.

[Bernardinis03] F. De Bernardinis, M. Jordan, and A. Sangiovanni Vincentelli. Support vector machines for analog circuit performance representation. *Proc. Design Automation Conf.*, pages 964–969, 2003.

[Bernardinis04] F. De Bernardinis, S. Gambini, F. Vincis, F. Svelto, R. Castello, and A. Sangiovanni-Vincentelli. Design space exploration for a UMTS front end exploiting analog platforms. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 923–930, 2004.

[Bernardinis05a] F. De Bernardinis, P. Nuzzo, and A. Sangiovanni-Vincentelli. Mixed signal design space exploration through analog platforms. *Proc. Design Automation Conf.*, pages 875–880, 2005.

[Bernardinis05b] F. De Bernardinis and A. Sangiovanni Vincentelli. Efficient analog platform characterization through analog constraint graphs. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 415–421, 2005.

[Binkley03] D. M. Binkley, C. E. Hopper, S. D. Tucker, B. C. Moss, J. M. Rochelle, and D. P. Foty. A CAD methodology for optimizing transistor current and sizing

in analog CMOS design. *IEEE Trans. Computer-Aided Design*, 22(2):225–237, February 2003.

[Boser92] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. *Proc. 5th Annu. ACM Workshop COLT, D. Haussler, Ed.*, pages 144–152, 1992.

[Bourguet04] V. Bourguet, L. de Lamarre, and M. M. Louërat. A layout-educated analog design flow. *The 47^{th} IEEE Midwest Symposium on Circuits And Systems*, 1(I):485–488, July 2004.

[Brayton81] R. Brayton, D. Hachtel, and A. Sangiovanni-Vincentelli. A survey of optimization techniques for integrated circuits. *IEEE Press*, 69(10):1334–1362, 1981.

[Brooks99] B. Brooks. Standardizing compact models for IC simulation. *IEEE Circuits and Devices Magazine*, 15(4):10–13, July 1999.

[Carley90] Richard L. Carley. Automated design of operational amplifires: A case study. In Mohammed Ismail and José Franca, editors, *Introduction to Analog VLSI Design Automation*, pages 45–78. Kluwer Academic Publishers, 1990.

[Carley93] L. R. Carley, P. C. Maulik, E. S. Ochotta, and R. A. Rutenbar. *Analog Cell-Level Synthesis Using a Novel Problem Formulation. Advances in Analog Circuit Design*. Kluwer Academic Publishers, 1993.

[Chamla05] David Chamla, Andreas Kaiser, Andreia Cathelin, and Didier Belot. A G_m-C low-pass filter for zero-IF mobile applications with very wide tuning range. *IEEE J. of Solid-State Circuits*, 40(7):1443–1450, July 2005.

[Chan87] T. Y. Chan, J. Chen, P. K. Ko, and C. Hu. Impact of gate-induced dain leakage current on device scaling. *IEDM Tech. Dig.*, pages 718–721, 1987.

[Chang97] H. Chang, E. Charbon, U. Choudhury, A. Casotto, and A. Sangiovanni-Vincentelli. *A top-down constrained-driven design methodology for analog integrated circuits*. Kluwer Academic Publishers, 1997.

[Chatterjee05] S. Chatterjee, Y. Tsividis, and P. Kinget. 0.5-V analog circuit techniques and their application in OTA and filter design. *IEEE J. of Solid-State Circuits*, 40(12):2373– 2387, December 2005.

[Cheng99] Y. Cheng and C. Hu. *MOSFET Modeling and BSIM3 User's Guide*. Kluwer Academic Publishers, 1999.

[Ciranova] Ciranova. *http://www.ciranova.com/*.

[Coello00] Carlos A. Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.

[Coughran83] W. M. Coughran, E. Grosse, and D. J. Rose. CAzM: A circuit analyzer with macromodeling. *IEEE Trans. on Electron Devices*, 30(9):1207– 1213, September 1983.

[de Lamarre02] L. de Lamarre. *CAIRO+ : Intégration du modèle BSIM3V3, stage de DEA*. Université Pierre et Marie Curie, Lab. LIP6, dept. SOC, Paris, 2002.

[DeGrauwe84a] M. DeGrauwe and W. M. Sansen. A synthesis program for operational amplifiers. *Proc. IEEE Int. Solid State Circuit Conf.*, pages 18–19, 1984.

[DeGrauwe84b] M. G. R. DeGrauwe and W. M. C. Sansen. The current efficiency of MOS transconductance amplifiers. *IEEE J. of Solid-State Circuits*, SC-19(3), June 1984.

[DeGrauwe87a] M. G. DeGrauwe and *et al*. An Analog Expert Design System. *Proc. IEEE Int. Solid State Circuit Conf.*, 1987.

[DeGrauwe87b] M. G. R. DeGrauwe, E. Dijkstra O. Nys, J. Rijmenants, S. Bitz, B. L. A. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. van der Stappen, and H. J. Oguey. IDAC: An Interactive Design Tool for Analog CMOS Circuits. *IEEE J. of Solid-State Circuits*, SC-22(6):1106–1116, December 1987.

[Degrauwe87c] Marc G. R. Degrauwe and *et. al.* IDAC: An Interactive Design Tool for Analog CMOS Circuits. *IEEE J. of Solid-State Circuits*, 22(6):1106–1115, December 1987.

[DeGrauwe89] M. G. R. DeGrauwe, B. L. A. G. Goffart, C. Meixenberger, M. L. A. Pierre, J. B. Litsios, J. Rijmenants, O. J. A. P. Nys, E. Dijkstra, B. Joss, M. K. C. M. Meyvaert, T. R. Schwarz, and M. D. Pardeon. Towards an analog system design environment. *IEEE J. of Solid-State Circuits*, SC-24(3):659–672, June 1989.

[Dessouky01] M. Dessouky. *Design for Reuse of Analog Circuits. Case Study : Very Low-Voltage* $\Delta\Sigma$ *Modulator*. PhD thesis, Université Pierre et Marie Curie, Lab. LIP6, Paris, 2001.

[Doboli03] A. Doboli and R. Vemuri. Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS. *IEEE Trans. Computer-Aided Design*, 22(11):1504–1520, November 2003.

[Duvvury86] C. Duvvury. A guide to short channel effects in MOSFETs. *IEEE Circuits and Systems Magazine*, page 6, 1986.

[Foty97] D. Foty. *MOSFET modelling with SPICE, Principales and Practices*. Prentice-Hall, 1997.

[Fremuth-Paeger] C. Fremuth-Paeger. *http://www.math.uni-augsburg.de/ fremuth/goblin.html*.

[Galup-Montoro00] C. Galup-Montoro and M. C. Schneider. Resizing rules for the reuse of MOS analog design. *Proc. of the* 15^{th} *Symposium on integrated circuits and Systems Design*, pages 89–93, September 2000.

[Galup-Montoro02] C. Galup-Montoro, M. C. Schneider, and R. M. Coitinho. Resizing rules for MOS analog-design reuse. *IEEE Design and Test of Computers*, 19(2):50–58, March-April 2002.

[Gielen89] G. E. Gielen, H. C. C. Walscharts, and W. M. C. Sansen. ISAAC: A symbolic simulator for analog circuits. *IEEE J. of Solid-State Circuits*, 24(6):1587–1597, December 1989.

[Gielen90a] G. E. Gielen. *Design Automation for Analogue Integrated Circuits*. PhD thesis, Katholieke Universiteit Leuven, October 1990.

[Gielen90b] G. E. Gielen, H. C. Walscharts, and W. C. Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE J. of Solid-State Circuits*, 25(3):707–713, June 1990.

[Gielen91] G. E. Gielen and W. Sansen. *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Kluwer Academic Publishers, 1991.

[Gielen93] George Gielen, Koen Swings, and Willy Sansen. Open analog synthesis system based on declarative methods. In Johan H. Huijsing, Rudy J. van der Plassche, and Willy Sansen, editors, *Analog Circuit Design: Operational Amplifiers, Analog to Digital Converters and Analog Computer Aided Design*, pages 421–445. Kluwer Academic Publishers, 1993.

[Gielen94] G. Gielen, P. Wambacq, and W. Sansen. Symbolic analysis methods and applications for analog circuits: A turotial overview. *IEEE Press*, 82(2):287–304, February 1994.

[Gielen95] Georges Gielen, Geert Debyser, Piet Wambacq, Koen Swings, and Willy Sansen. Use of symbolic analysis in analog circuit synthesis. In *?*, pages 2205–2208, February 1995.

[Gielen00] G. Gielen and R. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *IEEE Press*, 88(12):1825–1854, December 2000.

[Graeb01] H. Graeb, S. Zizala, J. Eckmueller, and K. Antreich. The sizing rules method for analog integrated circuit design. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 343–349, November 2001.

[Guindi95] R. S. Guindi and M. I. Elmasry. High-level analog synthesis using signal flow graph transformations. *IEEE Press*, pages 366–369, September 1995.

[Hamour03] A. Hamour, R. Saleh, S. Mirabbasi, and A. Ivanov. Analog IP design flow for SoC applications. *Proc. IEEE Int. Symposium on Circuits and Systems*, IV:676–679, 2003.

[Hanafi93] H. Hanafi. A model for anomalous short channel behavior in MOSFETs. *IEEE Electron Device Letters*, EDL-14:575, 1993.

[Harjani87] R. Harjani, R. A. Rutenbar, and L. R. Carley. A prototype framework for knowledge-based analog circuit synthesis. *Proc. Design Automation Conf.*, pages 42–49, June 1987.

[Harjani88] R. Harjani, R. A. Rutenbar, and L. R. Carley. Analog circuit synthesis for performance in oasys. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, November 1988.

[Harjani89a] R. Harjani. *OASYS: A Framework for Analog Circuit Synthesis*. PhD thesis, Carnegie Mellon University, Pittsburgh PA, March 1989.

[Harjani89b] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: A framework for analog circuit synthesis. *IEEE Trans. Computer-Aided Design*, 8(12):1247–1266, December 1989.

[Harvey92] J. P. Harvey, M. I. Elmasry, and B. Leung. STAIC: An interactive framework for synthesizing CMOS and BiCMOS analog circuits. *IEEE Trans. Computer-Aided Design*, CAD-11(11):1402–1418, November 1992.

[Ho83] C. W. Ho, A. E. Ruehli, and P. A. Brennan. The modified nodal approach to network analysis. *IEEE Trans. on Circuits and Systems*, 22(6):504–509, June 1983.

[Hsueh88] K. K. Hsueh, J. J. Sanchez, T. A. Demassa, and L. A. Akers. Inverse-narrow-width effects and small geometry MOSFET threshold voltage model. *IEEE Trans. on Electron Devices*, ED-35:325–338, 1988.

[Hu05] B. Hu, C. Wakayama, L. Zhou, and C.-J. R. Shi. Developing device models. *IEEE Circuits and Devices Magazine*, 21(4):6–11, July 2005.

[Iskander03] Ramy Iskander, Mohamed Dessouky, Maie Aly, Mahmoud Magdy, Noha Hassan, Noha Soliman, and Sami Moussa. Synthesis of CMOS analog cells using AMIGO. *Proc. Design Automation and Test in Europe Conf., Designer's Forum*, 2(2):20297–20302, March 2003.

[Iskander04] Ramy Iskander, Laurent de Lamarre, Andreas Kaiser, and Marie-Minerve Louërat. Design space exploration for analog IPs using CAIRO+. *Proc. of the IEEE Conf. on Electrical, Electronic and Computer Engineering*, pages 473–476, September 2004.

[Iskander08] R. Iskander, M. M. Louërat, and A. Kaiser. Automatic DC operating point computation and design plan generation for analog IPs. *Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers*, 56(1-2):93–105, Institut für Mathematik,Lehrstuhl für Diskrete Mathematik, Optimierung und Operations Research,Universität Augsburg 2008.

[Jancke06] R. Jancke and P. Schwarz. Supporting analog synthesis by abstracting circuit behavior using a modeling methodology. *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 1471–1474, May 2006.

[Ji83] C. R. Ji and C.T. Sah. Analysis of the narrow gate effect in submicrometer MOSFET's. *IEEE Trans. on Electron Devices*, ED-30:1672–1677, 1983.

[Ju91] Y. C Ju, V. Rao, and R. Saleh. Consistency checking and optimization of macromodels. *IEEE Trans. Computer-Aided Design*, 10(8):957–967, Institut für Mathematik,Lehrstuhl für Diskrete Mathematik, Optimierung und Operations Research,Universität Augsburg 1991.

[Kayal06] M. Kayal and M. Blagojevic. Design methodology based on the analog blocks retargeting from bulk to FD SOI using ekv model. *Proc. of the Int. Conf. on Mixed Design*, pages 131–135, June 2006.

[Keating02] M. Keating and P. Bricaud. *Reuse Methodology Manual: For System-on-a-Chip Designs*. Kluwer Academic Publishers, third edition, 2002.

[Keutzer00] K. Keutzer, S. Malik, R. Newton, J. Rabaey, , and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. Computer-Aided Design*, 19(12):1523–1543, December 2000.

[Koh87] Han Young Koh, Carlo H. Sequin, and Paul R. Gray. Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 502–505, November 1987.

[Koh90] Han Young Koh, Carlo H. Sequin, and Paul R. Gray. OPASYN: A Compiler for CMOS Operational Amplifiers. *IEEE Trans. Computer-Aided Design*, 9(2):113–125, February 1990.

[Krasnicki99] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley. MAELSTROM: Efficient simulation-based synthesis for custom analog cells. *Proc. Design Automation Conf.*, pages 945–950, June 1999.

[Krasnicki01] M. J. Krasnicki, R. Phelps, J. R. Hellums, M. McClung, R. A. Rutenbar, and L. R. Carley. ASF: A practical simulation-based methodology for the synthesis of custom analog circuits. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 350–357, 2001.

[Kundert00] K. Kundert, H. Chang, D. Jeffries, G. Lamant, E. Malasavi, and F. Sendig. Design of mixed-signal systems-on-a-chip. *IEEE Trans. Computer-Aided Design*, 19(12):1561–1571, December 2000.

[Kung85] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI and Modern Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985.

[Lagarias98] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.

[Lee74] A. Y. Lee. Signal flow graphs: Computer-aided system analysis and sensitivity calculations. *IEEE Trans. on Circuits and Systems*, CAS-21(2):209–216, March 1974.

[Levi07a] T. Levi, J. Tomas, N. lewis, and P. Fouillat. Resizing methodology for cmos analog circuit. *Proc. of SPIE*, 6590, May 2007.

[Levi07b] T. Levi, J. Tomas, N. lewisa, and P. Fouillat. IP-based design reuse for analogue systems: a case study. *Proc. of SPIE*, 6590, May 2007.

[Leyn98] F. Leyn, G. Gielen, and W. Sansen. An efficient DC root solving algorithm with guaranteed convergence for analog integrated CMOS circuits. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 304–307, November 1998.

[Li03] Z. Li, L. Luo, and J. Yuan. A study on analog IP blocks for mixed-signal SoC. *IEEE Int. Conf. on ASIC*, 1:564–567, 2003.

[Liu02] H. Liu, A. Singhee, R. Rutenbar, , and L. Carley. Remembrance of circuit past: Macromodeling by data mining in large analog design spaces. *Proc. Design Automation Conf.*, pages 437–442, 2002.

[Lopez04] R. C. Lopez. *Metodologías y Herramientas de Reusabilidad en el Diseño de Circuitos Integrados Analógicos y de Señal Mixta*. PhD thesis, Universidad de Sevilla, December 2004.

[Lopez05] R. Castro Lopez, F. V. Fernandez, and A. R. Vazquez. A reuse-based framework for the design of analog and mixed-signal ICs. *Proc. of SPIE*, 5837(3):25–36, May 2005.

[Madrid01] N. Madrid, E. Peralias, A. Acosta, and A. Rueda. Analog/mixed-signal IP modeling for design reuse. *Proc. Design Automation and Test in Europe Conf.*, pages 766–767, March 2001.

[Makris92] C. A. Makris and C. Toumazou. Qualitative reasoning in analog IC design automation. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 8.3/1–4, May 1992.

[Makris95] C. A. Makris and C. Toumazou. Analog IC design automation: Part II - automated circuit correction by qualitative reasoning. *IEEE Trans. Computer-Aided Design*, 14(2):239–254, February 1995.

[Martens08] Ewout S. J. Martens and Georges G. E. Gielen. *High-Level Modeling and Synthesis of Analog Integrated Systems*. Analog Circuits and Signal Processing, Springer, 2008.

[Mason56] Samuel J. Mason. Feedback theory-further properties of signal flow graphs. *Proc. of the I.R.E.*, 44(7):920–926, July 1956.

[MathWorks] The MathWorks. *http://www.mathworks.com*.

[Maulik91] P. C. Maulik and L. R. Carley. Automating analog circuit design using constrained optimization techniques. *Proc. IEEE Int. Conf. on Computer-Aided-Design*, pages 390–393, November 1991.

[Maulik92a] P. C. Maulik. *Formulations for Optimization-based Synthesis of Analog Cells*. PhD thesis, Carnegie Mellon University, Pittsburgh PA, September 1992.

[Maulik92b] P. C. Maulik, L. R. Carley, and R. A. Rutenbar. A mixed integer non-linear programming approach to analog circuit synthesis. *Proc. Design Automation Conf.*, pages 698–703, June 1992.

[Maulik93] P. C. Maulik, L. R. Carley, and D. J. Allstot. Sizing of cell-level analog circuits using constrained optimization techniques. *IEEE J. of Solid-State Circuits*, 28(3):233–241, March 1993.

[McConaghy05] T. McConaghy, T. Eecklaert, , and G. Gielen. CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming. *Proc. Design Automation and Test in Europe Conf.*, pages 1082–1087, 2005.

[MOS-AK] MOS-AK. *http://www.mos-ak.org/*.

[Nelder65] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[Nishida81] M. Nishida and H. Onodera. An anomalous increase of threshold voltage with shortening of the channel lengths fo deeply boron-implanted n-channel MOSFETs. *IEEE Trans. on Electron Devices*, ED-28:1101, 1981.

[Ochotta96] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley. Synthesis of high-performance analog circuits in ASTRX/OBLX. *IEEE Trans. Computer-Aided Design*, 15(3):273–294, March 1996.

[Ochotta98] Emil S. Ochotta, Tamal Mukherjee, Rob A. Rutenbar, and L. Richard Carley. *Practical Synthesis of High-Performace Analog Circuits*. Kluwer Academic Publishers, first edition, 1998.

[Phelps00] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, and J. R. Hellums. ANACONDA: Robust synthesis of analog circuits via stochastic pattern search. *IEEE Trans. Computer-Aided Design*, pages 567–570, 2000.

[Pillage95] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic Circuit and System Simulation Methods*. McGraw-Hill, 1995.

[Plas01] G. Van der Plas, G. Debyser, F. Leyn, K. Lampaert, J. Vandenbussche, G. Gielen, W. Sansen, P. Veselinovic, and D. Leenaerts. AMGIE-A synthesis environment for CMOS analog integrated circuits. *IEEE Trans. on Circuits and Systems*, 20(9):1037–1058, September 2001.

[Porte08] J. Porte, 2008. OCEANE, *http://www-asim.lip6.fr/recherche/oceane*.

[Rabaey06] J. M. Rabaey, F. De Bernardinis, A. M. Niknejad, B. Nikolic, and A. Sangiovanni-Vincentelli. Embedding mixed-signal design in systems-on-chip. *IEEE Press*, 94(6):1070–1088, June 2006.

[Rajsuman00] R. Rajsuman. *System-on-Chip Design and Test*. Kluwer Academic Publishers, 2000.

[Rashinkar01] P. Rashinkar, P. Paterson, and L. Singh. *System-on-a-Chip Verification*. Kluwer Academic Publishers, 2001.

[Richardson89] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. *Proc. 3rd Int. Conf. on Genetic Algorithms*, pages 191–197, June 1989.

[Roychowdury04] J. Roychowdury. An overview of automated macromodeling techniques for mixed-signal systems. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 109–116, 2004.

[Sagantec] Sagantec. *http://www.sagantec.com/*.

[Saleh06] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Memieux, P. Pande, C. Grecu, and A. Ivanov. System-on-chip: Reuse and integration. *IEEE Press*, 94(6):1050–1069, June 2006.

[Savio04] A. Savio, L. Colalongo, , Z. M. Kovacs-Vajna, and M. Quarantelli. Scaling rules and parameter tuning procedure for analog design reuse in technology migration. *Proc. IEEE Int. Symposium on Circuits and Systems*, 5:V 117–V 120, May 2004.

[Savio06] A. Savio, L. Colalongo, M. Quarantelli, and Z. M. Kovacs-Vajna. Automatic scaling procedures for analog design reuse. *IEEE Trans. on Circuits and Systems*, 53(12):2539–2547, December 2006.

[Sedra91] Adel S. Sedra and Kenneth C. Smith. *Microelectronic Circuits*. Hold, Rinehart and Winston, third edition, 1991.

[Semiconductors] International Technology Roadmap for Semiconductors. *http://www.itrs.net*.

[SIA98] SIA. *SIA Roadmap update*, 1998.

[Silveira96] F. Silveira, D. Flandre, and P. G. A. Jespers. A g_m/I_D based methodology for the design of CMOS analog circuits and its application to the synthesis of a silicon-on-insulator micropower OTA. *IEEE J. of Solid-State Circuits*, 31(9):1314–1319, September 1996.

[Simucad] Simucad. *http://www.simucad.com/*.

[Smith96] S. Smith and E. Sanchez-Sinencio. Low voltage integrators for high-frequency CMOS filters using current mode techniques. *IEEE Trans. on Circuits and Systems*, 43(1):39–48, January 1996.

[Spence99] Robert Spence. The facilitation of insight for analog design. *IEEE Trans. Computer-Aided Design*, 46(5):540–548, May 1999.

[Starzyk86] J. A. Starzyk and A. Konczykowska. Flowgraph analysis of large electronic networks. *IEEE Trans. on Circuits and Systems*, CAS-33(3):302–315, March 1986.

[Stefanovic03] D. Stefanovic, M. Kayal, M. Pastre, and V. B. Litovski. Procedural analog design (PAD) tool. *Proc. of the 4th Int. Symposium on Quality Electronic Design*, pages 313–318, March 2003.

[Stefanovic05] D. Stefanovic, M. Kayal, and M. Pastre. PAD: A new interactive knowledge-based analog design approach. *Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers*, pages 291–299, March 2005.

[Stefanovic07] D. Stefanovic, S. Pesenti, M. Pastre, and M. Kayal. Structured design based on the inversion factor parameter: Case study of $\Delta\Sigma$ modulator system. *Proc. of the 14th Int. Conf. on Mixed Design*, pages 95–102, June 2007.

[Swings91a] K. Swings, S. Donnay, and W. Sansen. HECTOR: A hierarchical topology-construction program for analog circuits based on a declarative approach to circuit modelling. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 5.3/1–5, 1991.

[Swings91b] K. Swings, G. Gielen, and W. Sansen. An intelligent analog IC design system based on manipulation of design equations. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 8.6/1–5, 1991.

[Swings91c] Koen Swings and Willy Sansen. DONALD: A workbench for interactive design space exploration and sizing of analog circuits. In *Proc. European Design Automation Conf.*, pages 475–479, 1991.

[SystemC-AMS] SystemC-AMS. *http://www.systemc-ams.org/*.

[Tiernan70] J. C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM*, 13(12):722–726, December 1970.

[Toumazou90] C. Toumazou, C. A. Makris, and C. M. Berrah. ISAID- a methodology for automated analog IC design. *Int. Symp. on Circuits and Systems*, pages 531–533, 1990.

[Tuong06] P. N. Tuong. *Définition et Implémentation d'un Langage de Conception de Composants Analogiques Réutilisables*. PhD thesis, Université Pierre et Marie Curie, Lab. LIP6, Paris, 2006.

[Verilog-AMS] Verilog-AMS. *http://www.eda.org/verilog-ams/*.

[VHDL-AMS] VHDL-AMS. *http://www.vhdl.org/analog/*.

[Vlach94] J. Vlach and K. Singhal. *Computer methods for circuit analysis and design, Second Edition*. Van Nostrand Reinhold, 1994.

[Wambacq91] P. Wambacq, J. Vanthienen, G. Gielen, and W. Sansen. A design tool for weakly nonlinear analog integrated circuis with multiple inputs (mixers, multipliers). *Proc. IEEE Custom Integrated Circuits Conf.*, pages 5.1/1–5, 1991.

[Wambacq95] P. Wambacq, F. Fernandez, G. Gielen, W. Sansen, and A. Rodriguez-Vazquez. Efficient symbolic computation of approximated small-signal characteristics. *IEEE J. of Solid-State Circuits*, 30:327–330, March 1995.

[Wan03] B. Wan, B. P. Hu, L. Zhou, and C.-J. R. Shi. MCAST: an abstract-syntax-tree based model compiler for circuit simulation. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 249–252, September 2003.

[Watts06] J. Watts. Enhancing productivity by continuously improving standard compact models. *Proc. IEEE Custom Integrated Circuits Conf.*, pages 623–630, September 2006.

[Wu94] C.-H. Wu, S.-J. Lee, and H.-S. Chou. Dependency analysis for knowledge validation in rule-based expert systems. *Proc. of the 10th Conf. on Artificial Intelligence for Applications*, pages 327–333, March 1994.