Titre: Title:	Méthodes d'accélération de la simulation analogique utilisée dans des applications nécessitant des simulations multiples
Auteur: Author:	Michel Morneau
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Morneau, M. (2006). Méthodes d'accélération de la simulation analogique utilisée dans des applications nécessitant des simulations multiples [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7853/

UNIVERSITÉ DE MONTRÉAL

MÉTHODES D'ACCÉLÉRATION DE LA SIMULATION ANALOGIQUE UTILISÉE DANS DES APPLICATIONS NÉCESSITANT DES SIMULATIONS MULTIPLES

MICHEL MORNEAU
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU
DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
GÉNIE ÉLECTRIQUE
DÉCEMBRE 2006

Your file Votre référence
ISBN: 978-0-494-25562-9
Our file Notre référence
ISBN: 978-0-494-25562-9

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MÉTHODES D'ACCÉLÉRATION DE LA SIMULATION ANALOGIQUE UTILISÉE DANS DES APPLICATIONS NÉCESSITANT DES SIMULATIONS MULTIPLES

présenté par: MORNEAU Michel

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. SAVARIA, Yvon, Ph.D., président

M. KHOUAS, Abdelhakim, Ph.D., membre et directeur de recherche

M. DAVID, Jean-Pierre, Ph.D., membre

Remerciements

Je tiens d'abord à remercier mon directeur de recherche, Abdelhakim Khouas, professeur à l'École Polytechnique de Montréal, de m'avoir proposé ce projet de recherche qui s'est avéré fort intéressant et stimulant. Ses nombreux conseils ainsi que sa disponibilité m'ont grandement aidé à mener à terme mon projet et à acquérir une précieuse expérience en recherche.

Merci à mes collègues de bureau au GRM, avec qui j'ai eu l'occasion de discuter régulièrement de choses et d'autres et qui faisaient ainsi en sorte que mes journées de travail paraissaient plus courtes.

Je voudrais remercier du fond de mon cœur ma copine, Christine Pageau, qui m'a toujours supporté de façon inconditionnelle et désintéressée, même si elle a eu à subir mes absences fréquentes pour terminer ma maîtrise tout en occupant un emploi. Elle a fait en sorte que je pouvais décrocher et passer des moments merveilleux tout au long de ma maîtrise.

Enfin, je tiens à remercier mes parents, Marie-Paule et Roland Morneau, pour leur soutien tant moral que financier tout au long de mes études. Je leur dédie donc ce mémoire.

Résumé

Les simulateurs de type SPICE jouent un rôle prépondérant au niveau de la conception des circuits analogiques, mais ne sont pas adaptés au cas de simulations répétitives du même circuit avec de légères modifications. Trois applications majeures requièrent de telles simulations multiples : la simulation de pannes analogiques, le dimensionnement automatique de circuits et l'analyse Monte Carlo. Ces opérations nécessitent un temps de calcul très élevé. Le but de ce projet de recherche consiste à développer des techniques permettant de diminuer le temps de calcul pour les simulations multiples en favorisant le partage d'information d'un circuit à l'autre ainsi qu'en permettant un compromis entre le temps de calcul et la précision des résultats de simulation.

L'algorithme itératif *Newton-Raphson*, qui est au cœur de la simulation de circuits analogiques, est très sensible à l'approximation initiale utilisée et converge après un certain nombre d'itérations vers la solution exacte. La première approche de réduction du temps de calcul explorée consiste en des méthodes d'arrêt des itérations avant la convergence de l'algorithme *Newton-Raphson* en simulation DC. Un compromis entre le temps de calcul et la précision des résultats de simulation est ainsi réalisé. Les deux méthodes proposées nécessitent une approximation initiale précise de la solution et se basent sur la métrique de variation relative maximale également suggérée dans ce mémoire. La première méthode consiste à augmenter la tolérance relative du simulateur SPICE jusqu'au niveau de précision désirée. La simulation se termine dès que cette précision est atteinte avec un faible risque d'erreur et le nombre d'itérations requis est réduit de 20 à 40 %. La seconde méthode s'applique aux applications qui comparent des circuits entre eux, notamment la simulation de pannes. Cette fois, la simulation DC d'un circuit défectueux est terminée dès que sa tension de sortie est suffisamment précise pour que la panne soit immédiatement classifiée comme détectée ou non. Le nombre

d'itérations est réduit d'un facteur 3 avec un très faible pourcentage de pannes mal classifiées.

La seconde approche explorée dans ce mémoire consiste à optimiser un simulateur SPICE existant pour effectuer rapidement les simulations multiples d'un même circuit avec de légères modifications. Le simulateur *MultiSPICE* résultant diminue le temps de calcul engendré par les simulations multiples grâce à 1) une connexion *socket* avec l'application appelante, 2) l'injection efficace des modifications dans le circuit nominal, 3) l'utilisation d'approximations précises des solutions et 4) l'utilisation de méthodes d'arrêt de la simulation DC avant la convergence. Des applications de simulation de pannes analogiques, de dimensionnement automatique de circuits et d'analyse Monte Carlo ont été développées afin d'évaluer les performances obtenues par *MultiSPICE*. En simulation DC, le simulateur proposé est en moyenne 85 fois plus rapide que le simulateur original pour des résultats de simulation identiques. Lorsque les méthodes d'arrêt des itérations avant la convergence sont utilisées, le gain en temps de calcul atteint alors 125 pour une très faible proportion de résultats erronés.

Abstract

Although SPICE-like simulators are essential tools for the design of analog circuits, they are not intended to simulate repeatedly the same circuit with slight modifications. Three major applications require such multiple simulations: analog fault simulation, automatic circuit sizing and Monte Carlo analysis. These operations are CPU time intensive. The objective of this research project is the development of techniques allowing the reduction of the CPU time required for multiple simulations by favoring the information sharing between circuits and allowing a tradeoff between CPU time and simulation results accuracy.

The *Newton-Raphson* algorithm, on which is based the simulation of analog circuits, is very sensitive to the initial solution approximation used and converges to the exact solution after a certain number of iterations. The first computation time reduction approach proposed is the development of methods to end the iterations before convergence of *Newton-Raphson* algorithm in DC simulation. A tradeoff is then achieved between the CPU time and the accuracy of the simulation results. The two proposed methods require an accurate initial solution approximation and are based on the worst relative variation metric also proposed in this thesis. The first method is to increase the relative tolerance of the SPICE simulator to the required simulation accuracy. The simulation is ended once this accuracy is reached with a low risk of erroneous results and the number of performed iterations is reduced by 20 to 40 %. The second method addresses the applications which need comparisons between circuits, among others analog fault simulation. The DC simulation of a considered faulty circuit is ended once its output voltage is accurate enough to immediately classify the fault as detected or not detected. The number of iterations is reduced by a factor of 3, with a low percentage of misclassified faults.

The second approach explored in this thesis is to optimize an existing SPICE simulator to rapidly perform multiple simulations of a circuit with slight modifications. The resulting *MultiSPICE* simulator decreases the computation time required by multiple simulations by taking advantage of 1) a socket connection with the calling application, 2) efficient injection of modifications in nominal circuit, 3) the use of accurate initial DC solution approximations and 4) the use of methods to end DC simulation before convergence. Analog fault simulation, automatic circuit sizing and Monte Carlo analysis applications have been developed to evaluate the performances achieved by *MultiSPICE*. In DC simulation, the proposed simulator is 85 times faster than the reference simulator for identical simulation results. When methods to end iterations before convergence are used, the computation time gain reaches 125 for a very low risk of erroneous results.

Table des matières

Liste des figures

Liste des tableaux

Liste des sigles et abréviations

abstol	Absolute Tolerance
AC	Alternating Current
ACS	Analog Circuit Sizing
AFS	Analog Fault Simulation
AIPS	Approximation Initiale Précise de la Solution
AISA	Accurate Initial Solution Approximation
AMS	Analog and Mixed-Signal
BJT	Bipolar Junction Transistor
CAO	Conception Assistée par Ordinateur
DC	Direct Current
IEEE	Institute of Electrical and Electronics Engineers
MC	Monte Carlo analysis
MNA	Modified Nodal Analysis
MOS	Metal Oxide Semiconductor
NR	Newton-Raphson
PCB	Printed Circuit Board
reltol	Relative Tolerance
σ	Écart-type
SoC	System on Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
TBSA	Threshold-Based Simulation Accuracy
μ	Moyenne
VarRelMax	Variation Relative Maximale
VHDL	Very high speed integrated circuit Hardware Description Language
WorstRelVar	Worst Relative Variation
x^k	Solution (vecteur de tensions de noeuds) à l'itération k

Liste des annexes

Introduction

Au cours de la dernière décennie, le développement des systèmes sur puces (SoC) qui permettent d'intégrer plusieurs applications dans un même circuit intégré a stimulé le marché des circuits analogiques et mixtes. Les SoC permettent d'améliorer les performances et de réduire les coûts de fabrication des systèmes qui, traditionnellement, sont réalisés sur des cartes imprimées (PCB). C'est pourquoi de plus en plus d'applications intègrent des parties analogiques sur des circuits numériques afin d'interagir avec le monde extérieur, ce qui a engendré une forte croissance de la demande de circuits mixtes, atteignant les 20 % annuellement pour un marché dépassant les 22 milliards de dollars [15].

Or, les outils d'aide à la conception (CAO) disponibles pour la conception des circuits analogiques sont beaucoup moins efficaces que ceux qui sont utilisés pour les circuits numériques. En effet, les deux seuls outils matures disponibles aux concepteurs sont les éditeurs de circuits (*schematics*) et les simulateurs analogiques. Le reste du travail est généralement fait à la main et repose principalement sur l'expérience du concepteur. Par conséquent, pour plusieurs circuits mixtes analogique-numérique, le coût de fabrication et de test de la partie analogique accapare une part importante du temps de conception et du coût total du circuit. Bien qu'elle occupe généralement une petite partie de la surface de la puce, la partie analogique est souvent responsable d'erreurs de conception nécessitant de coûteuses itérations de modifications [60].

L'objectif à long terme dans lequel s'inscrit ce projet est le développement d'outils de CAO pour la conception et le test des circuits analogiques et mixtes. Avec la complexité croissante des circuits analogiques et mixtes, il est nécessaire d'améliorer continuellement les outils CAO afin de soutenir la demande mondiale tant en productivité qu'en qualité. Une automatisation des tâches complexes à l'aide d'outils efficaces est

devenue une nécessité, d'où les nombreux projets de recherche menés depuis quelques années tant au niveau académique qu'industriel [2].

Dans le cadre de ce projet, nous visons de façon plus spécifique les applications de CAO nécessitant des simulations multiples du même circuit avec de légères modifications. Trois d'entre elles ont été ciblées. Premièrement, les outils de simulation de pannes ont pour but de simuler les effets des défauts physiques sur un circuit donné de façon à créer un ensemble de tests permettant de détecter ces défauts. Les ensembles de tests à appliquer à chaque puce peuvent alors être optimisés tout en maintenant la même efficacité, ce qui permet de réduire les coûts de production. Pour ce faire, les outils de simulation de pannes injectent une panne à l'intérieur du circuit et font appel à un simulateur pour connaître son effet sur le fonctionnement du circuit. Deuxièmement, le dimensionnement des circuits analogiques se fait le plus souvent de façon manuelle, avec la technique d'essais et erreurs, qui est fastidieuse et ne conduit pas toujours à la solution optimale. Récemment, des outils permettant d'effectuer le dimensionnement de circuits ont été développés. Ceux-ci recherchent les dimensions optimales des transistors permettant d'atteindre un compromis acceptable au niveau des spécifications. Pour ce faire, un algorithme d'optimisation génère des circuits candidats qui doivent être simulés afin de mesurer leurs performances. Troisièmement, l'analyse Monte Carlo permet de prédire le comportement d'un circuit face aux imperfections du procédé de fabrication. À cette fin, les paramètres des composants du circuit sont soumis à des variations aléatoires suivant une distribution donnée, de façon à modéliser les effets des fluctuations du procédé de fabrication sur le comportement des puces fabriquées. Un grand nombre d'échantillons du circuit doit donc être simulé.

Ces trois applications importantes ont certains points en commun. D'une part, elles nécessitent un grand nombre de simulations d'un même circuit soumis à des variations de paramètres de composants. Le comportement des divers circuits modifiés est généralement très semblable. D'autre part, les résultats de simulation servent

principalement à une prise de décision pour chaque circuit modifié. Dans le cas de la simulation de pannes, on désire savoir si une panne est détectable ou non; pour le dimensionnement automatique de circuits, on veut comparer les performances entre les circuits candidats; et au niveau de l'analyse Monte Carlo, on recherche la robustesse de la conception et le rendement de fabrication en analysant le pourcentage de circuits qui respectent les spécifications. Comme on le verra plus loin, les simulateurs commerciaux, tels *Spectre* [6], *HSPICE* [4] et *Eldo* [40] ne sont pas conçus pour être appelés des milliers de fois pour un même circuit avec de légères variations.

Le but de ce projet de recherche consiste à développer des techniques permettant de diminuer le temps de calcul dans le cas des simulations multiples. La similitude entre les résultats des circuits modifiés ainsi que la possibilité de réduire la précision de ceux-ci sont donc exploités afin d'accélérer la simulation. À cette fin, deux approches ont été explorées :

1) Recherche de tests d'arrêt permettant de terminer la simulation plus rapidement en ajustant la précision du simulateur selon les besoins de l'application en vue d'obtenir une réduction du temps de calcul.

2) Développement d'un simulateur optimisé pour les simulations multiples ainsi que des applications exploitant les propriétés de ce simulateur.

Le présent mémoire se divise en six chapitres principaux et un très court chapitre. Tout d'abord, le premier chapitre débute par une introduction à la simulation de circuits analogiques à l'aide de simulateurs de type SPICE, en mettant l'accent sur l'algorithme itératif *Newton-Raphson* (NR) en simulation DC. Il présente ensuite les trois applications ciblées par ce projet ainsi que les méthodes de simulation existantes permettant de réduire le temps de calcul. L'accent est mis sur les méthodes de simulation DC faisant appel à un simulateur de type SPICE.

Le chapitre 2 étudie la convergence de l'algorithme NR durant la simulation DC et propose la métrique de variation relative maximale, qui sera réutilisée au chapitre 4. Un critère permettant l'arrêt de la simulation une fois que la précision spécifiée au simulateur est atteinte est présenté.

La mise en contexte des deux articles proposés dans le présent mémoire est présentée au chapitre 3. Pour sa part, le quatrième chapitre est constitué d'un article accepté pour publication dans la revue « *Journal of Electronic Testing: Theory and Applications* » et qui s'applique à la simulation de pannes en analyse DC. On y retrouve une méthode permettant de terminer la simulation une fois que les résultats intermédiaires permettent de classifier correctement une panne par rapport aux seuils de détection.

Le cinquième chapitre est lui aussi un article, soumis cette fois à la revue « *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* ». Il présente *MultiSPICE*, un simulateur optimisé pour effectuer rapidement les simulations multiples, plus particulièrement en analyse DC. Ses fonctionnalités sont exploitées par des applications de simulation de pannes, de dimensionnement automatique de circuits et d'analyse Monte Carlo, également développées dans le cadre de ce mémoire.

Une discussion générale portant sur l'ensemble des méthodes de simulation proposées dans le cadre de ce mémoire est présentée au chapitre 6.

Le dernier chapitre conclut le présent mémoire en faisant la synthèse des performances obtenues pour les méthodes suggérées et en apportant quelques suggestions de travaux futurs.

Enfin, en annexe se retrouvent des détails supplémentaires sur la simulation de circuits analogiques, un guide d'utilisation du simulateur optimisé *MultiSPICE* ainsi qu'une description des applications développées.

CHAPITRE 1

Revue de la littérature

1.1 Introduction

Dans ce mémoire, les applications de simulations multiples sont définies comme étant des applications qui effectuent des modifications sur un circuit nominal et qui font appel à un simulateur pour simuler chacun des circuits ainsi générés. Ces circuits, qui ressemblent au circuit initial, sont appelés circuits modifiés. Ce chapitre présente les applications de simulations multiples qui ont été ciblées, à savoir la simulation de pannes, les outils de dimensionnement de circuits et l'analyse Monte Carlo.

Tout d'abord, les notions de base en simulation de circuits analogiques nécessaires à la compréhension de la suite de ce mémoire sont abordées. Ensuite, une revue de la littérature pour chacune des applications ciblées est présentée. Bien qu'elle expose brièvement chacune des méthodes de simulation utilisées, cette revue insiste plus particulièrement sur les méthodes qui font appel à la simulation de type SPICE, au niveau transistor. La simulation de pannes est toutefois vue plus en profondeur, puisqu'une méthode s'appliquant particulièrement bien à cette application est proposée dans ce mémoire.

Par la suite, certaines caractéristiques communes à ces applications sont ressorties. Ces caractéristiques servent d'ailleurs de base aux diverses méthodes de réduction du temps de simulation qui sont suggérées dans les chapitres subséquents.

1.2 Notions de base en simulation analogique

Bien que plusieurs simulateurs de circuits analogiques existent sur le marché, leur fonctionnement est très semblable puisqu'ils sont basés sur le simulateur SPICE développé à la fin des années 60 [62]. Cette section résume leur fonctionnement général, c'est-à-dire la représentation du circuit en mémoire, la résolution des systèmes d'équations non linéaires ainsi que la complexité algorithmique de la simulation de circuits. Le développement beaucoup plus détaillé des notions de simulations analogiques se retrouve à l'annexe A.

Tout d'abord, un simulateur doit lire la description du circuit, qui est habituellement dans un format *netlist* [49]. Chaque composant du circuit ainsi que ses connexions est décrit par une ligne dans le fichier *netlist*. Le circuit est par la suite représenté à l'intérieur d'un système matriciel. Pour ce faire, la loi des nœuds est généralement utilisée, c'est-à-dire que les conductances entre chaque élément sont représentées dans une matrice alors que les sources de courant continues se retrouvent dans un vecteur. Son extension, la représentation *Modified Nodal Analysis* (MNA), permet de représenter les sources de tension indépendantes, les inductances et les sources de tension / courant dépendantes à l'aide de lignes et colonnes supplémentaires dans le système matriciel [61]. L'algorithme de *Crout* [14], qui utilise la décomposition LU et des substitutions avant et arrière, est habituellement utilisé pour résoudre le système matriciel obtenu. On obtient alors les tensions à chacun des nœuds du circuit. Plus de détails sur le système MNA se retrouvent en annexe (section A.2).

La présence de transistors, diodes et autres éléments décrits par plus d'une équation dans le circuit produit un système d'équations non linéaires. Les simulateurs de type SPICE utilisent l'algorithme itératif NR pour résoudre les systèmes d'équations non linéaires ainsi obtenus. L'algorithme démarre avec l'approximation initiale de la solution, x^0, qui contient la tension à chaque nœud du circuit. Les composants non linéaires du circuit sont

alors linéarisés selon les tensions de cette approximation, ce qui permet d'obtenir les conductances et les sources de tension / courant équivalentes entre chacun des nœuds auxquels est connecté chaque composant [29]. Il est à noter que les valeurs obtenues par la linéarisation ne sont valides que pour les tensions utilisées. On obtient alors une matrice de conductances ainsi qu'un vecteurs de sources de courant continues. Une fois ce système matriciel résolu, une nouvelle approximation de la solution, x^1, est obtenue. Celle-ci est utilisée à l'itération suivante afin de permettre une nouvelle linéarisation des composants non linéaires. Des itérations sont effectuées jusqu'à ce que l'algorithme NR converge vers la solution exacte. L'équation 1.1 illustre la $k^{ème}$ itération NR, pour laquelle $J(x^{k-1})$ représente la matrice jacobienne, c'est-à-dire les conductances linéarisées, x^{k-1} est l'approximation de la solution obtenue à l'itération précédente, x^k est la nouvelle approximation de la solution, c'est-à-dire l'inconnue, et F est une fonction à minimiser. J et F sont construites durant l'étape de linéarisation des composants à partir des équations encapsulées dans les modèles des composants.

$$J(x^{k-1})\,x^k = J(x^{k-1})\,x^{k-1} - F(x^{k-1}) \qquad (1.1)$$

Il est à noter que les composants linéaires, telles les résistances et les condensateurs, ont toujours des valeurs constantes dans la matrice jacobienne et, par conséquent, une seule itération NR est effectuée dans le cas de circuits linéaires. Pour les composants non linéaires, la convergence est atteinte lorsque les solutions obtenues pour deux itérations consécutives diffèrent par moins que la valeur de tolérance donnée. Comme montré à l'équation 1.2, la tolérance tol_i^k pour le nœud i à l'itération k dépend des tolérances absolue (*abstol*) et relative (*reltol*) et de l'approximation de la solution pour ce nœud aux itérations courante (x^k) et précédente (x^{k-1}). Le critère de convergence utilisé dans *SPICE3f5* [49] est montré à l'équation 1.3.

$$tol_i^{\,k} = abstol + reltol \times \max\left(|\,x_i^{\,k}\,|, |\,x_i^{\,k-1}\,|\right) \qquad (1.2)$$

$$\text{Si} \quad |\,x_i^{\,k} - x_i^{\,k-1}\,| \le tol_i^{\,k} \quad \forall\, i \quad \rightarrow \text{convergence atteinte} \qquad (1.3)$$

Les valeurs *abstol* et *reltol* sont des paramètres communs aux simulateurs SPICE, représentant respectivement une tolérance absolue et une tolérance relative. Les valeurs par défaut dans *Spectre* [6] et *SPICE3f5* [49] sont : *abstol* = 10^{-6} V pour les tensions et 10^{-9} A pour les courants, alors que *reltol* = 0.001. À moins d'avoir des valeurs de l'ordre de *abstol*, l'influence de *abstol* est négligeable par rapport à *reltol*. Si une tension de nœud varie par plus que sa tolérance entre deux itérations, telle que définie par l'équation 1.2, une nouvelle itération doit être effectuée. Lorsque l'algorithme NR converge selon ce critère, la solution finale x^k est considérée comme la solution exacte générée par le simulateur. Soit ε_i^k l'erreur d'approximation pour le nœud i à l'itération k, si la solution approximative x^k est près de la solution exacte, on a l'équation suivante :

$$\lim_{k\to\infty} \frac{\left|\varepsilon_i^{k+1}\right|}{\left|\varepsilon_i^{k}\right|^2} = constante \qquad (1.4)$$

L'équation 1.4 montre que l'algorithme NR converge de façon quadratique lorsque la solution approximative est près de la solution exacte [61]. Par exemple, si *constante* = 1, on peut s'attendre à avoir $\varepsilon_i^{k+1} = 0.01$ et $\varepsilon_i^{k+2} = 0.0001$ pour les itérations k+1 et k+2 si la solution x^k est près de la solution exacte. Par contre, si l'approximation x^k est éloignée de la solution exacte, rien ne peut être assumé et des itérations supplémentaires sont requises avant la convergence [7]. Si la convergence n'est pas atteinte après un certain nombre d'itérations, les simulateurs commerciaux font appel à des techniques dites de continuation afin de trouver la solution au terme d'un certain nombre d'étapes intermédiaires. Ces techniques exécutent l'algorithme NR plusieurs fois, en commençant par une version du circuit généralement plus facile à résoudre, puis en augmentant la complexité jusqu'au circuit nominal [39].

De façon générale, la complexité algorithmique de la simulation DC est représentée par l'équation 1.5, où N est le nombre d'itérations NR effectuées, t est le nombre de

transistors et autres composants non linéaires du circuit et n est la taille du système matriciel, alors que K_1 et K_2 sont des constantes [22].

$$Coût = N\left(K_1 t + K_2 n^{1.5}\right) \qquad (1.5)$$

Bien que la réduction de la complexité des modèles de transistors (K_1) et l'accélération de la résolution du système matriciel (K_2) contribuent à réduire le temps global de simulation, la réduction du nombre d'itérations effectuées est le paramètre qui est le plus susceptible de réduire de façon significative le temps de calcul, peu importe la taille du circuit. Des développements supplémentaires sur l'algorithme NR et ses propriétés de convergence se retrouvent en annexe (section A.3).

Jusqu'ici, nous avons décrit de quelle façon il est possible de trouver le point de polarisation DC d'un circuit non linéaire. Les autres types d'analyse de circuit sont réalisés de façon similaire. L'analyse de balayage DC (*DC sweep*) est considérée comme une série de points d'opération DC à effectuer de façon séquentielle pour chaque valeur attribuée au paramètre balayé. En analyse fréquentielle AC, le point de polarisation DC doit tout d'abord être trouvé, puis le système matriciel résultant est résolu pour chaque valeur de fréquence à simuler. Pour sa part, l'analyse transitoire est elle aussi une suite de points, mais cette fois une intégration numérique est effectuée sur chaque capacité et inductance du circuit afin d'actualiser la quantité d'énergie emmagasinée, selon les valeurs de tension et courant à leurs bornes au temps considéré. Encore une fois, plus d'informations sont données en annexe (section A.4). Un résumé utile se retrouve aussi dans [44].

1.3 Simulation de pannes

La simulation de pannes permet de définir les ensembles de tests à appliquer sur les circuits intégrés afin d'être en mesure de détecter la grande majorité des défauts de

fabrication. L'objectif est de maximiser la proportion des défauts détectés, c'est-à-dire la couverture de pannes, tout en minimisant le temps de test pour chaque circuit. Les défauts de fabrication sont modélisés par des pannes en utilisant des modèles de pannes. En injectant des pannes dans le circuit nominal et en simulant chaque circuit défectueux ainsi obtenu, il est possible de déterminer à l'avance les pannes détectées par chaque test. Le schéma général de la simulation de pannes est le suivant :

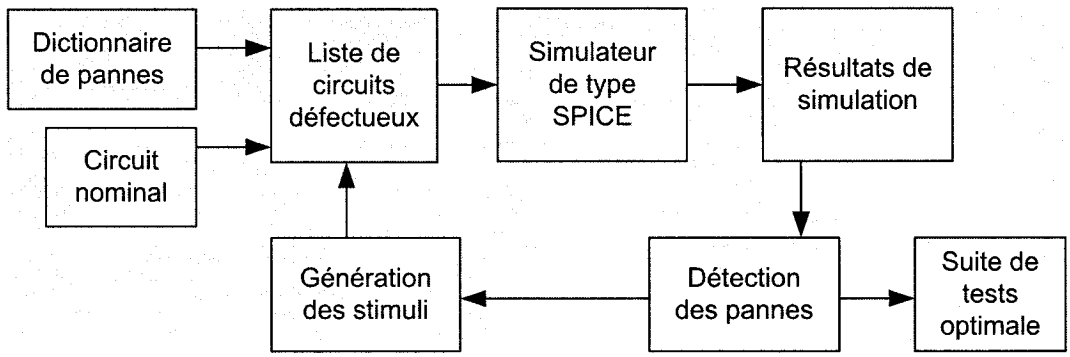


Figure 1.1 : Schéma-bloc de la simulation de pannes analogiques

Tout d'abord, le *netlist* du circuit nominal ainsi qu'un dictionnaire contenant les types de pannes à injecter doivent être fournis. Par la suite, une liste de circuits défectueux est générée à partir des modèles de pannes et le stimulus est défini. Une fois tous les circuits simulés, les résultats sont analysés afin de déterminer quelles pannes sont détectées à partir de ces stimuli. De nouveaux stimuli sont par la suite déterminés afin d'effectuer de nouvelles itérations avec toutes les pannes ou seulement celles qui sont non détectées jusqu'ici, selon les besoins. À la fin, on se retrouve avec un ensemble de tests optimal qui offre la couverture de pannes maximale.

1.3.1 Simulation et détection des pannes

Les pannes qui altèrent de façon importante le circuit nominal peuvent généralement être détectées par une simulation DC [21]. Par contre, d'autres pannes ne sont détectables qu'à haute fréquence et nécessitent par conséquent une simulation AC [23]. Enfin,

certaines pannes ne sont visibles que par une simulation transitoire puisqu'elles affectent des caractéristiques temporelles du circuit [22]. Or, la simulation DC est plus rapide que la simulation AC, qui est beaucoup plus rapide que la simulation transitoire. Par conséquent, il est souhaitable de détecter le plus de pannes possible en simulation DC.

Chaque stimulus détecte un certain nombre de pannes, alors qu'une panne peut être détectée par une certaine proportion des stimuli. Une panne est considérée comme détectable sous un stimulus donné si sa réponse diffère suffisamment de celle du circuit nominal. Trois méthodes sont généralement utilisées pour déterminer les seuils de détection d'une panne à une sortie donnée :

1) Valeur de déviation absolue (par exemple, ±50 mV) : Les pannes pour lesquelles la sortie diffère par plus qu'une certaine tension par rapport au circuit nominal sont classées comme détectées.

2) Pourcentage de la valeur de sortie du circuit nominal (par exemple, ±5 %) : Les pannes qui modifient la sortie par plus qu'un pourcentage donné de sa valeur nominale sont classées comme détectées.

3) Facteur de la distribution du circuit nominal à sa sortie (par exemple, ±2*l'écart type de la sortie du circuit nominal) : Les pannes qui modifient la sortie par plus d'un certain nombre de fois l'écart-type de la distribution du circuit nominal sont classées comme détectées.

Les deux premières méthodes sont simples puisque les seuils peuvent être calculés immédiatement après la simulation du circuit nominal. Si la réponse du circuit défectueux n'est pas à l'intérieur de l'intervalle, la panne est considérée détectée. Par contre, les fluctuations du procédé de fabrication peuvent masquer les effets de certaines pannes, ce qui limite la validité de ces méthodes [56]. Afin de considérer les fluctuations aléatoires dues au procédé de fabrication, la troisième méthode doit être utilisée. Des simulations Monte Carlo sont alors requises pour calculer les distributions du circuit nominal (D_n) et des circuits défectueux (D_f) à la sortie considérée. Pour une distribution gaussienne, la

probabilité qu'une variable aléatoire V aie une valeur à l'intérieur d'un intervalle est donnée par l'aire sous la courbe de la distribution dans cet intervalle. Si μ est la moyenne et σ l'écart-type, cette probabilité est de 95 % dans l'intervalle $[\mu-2\sigma, \mu+2\sigma]$. S'il n'y a pas de chevauchement entre les distributions D_n et D_f, la panne f est considérée détectable; si D_f est incluse dans D_n, la panne n'est pas détectable; sinon, la panne est partiellement détectable. L'équation 1.6 illustre le critère de détection considérant les variations dues au procédé de fabrication :

$$|\mu_n - \mu_f| > 2\sigma_n + 2\sigma_f \quad \rightarrow \text{la panne est détectée} \qquad (1.6)$$

Dans cette équation, μ_n et μ_f représentent respectivement la moyenne du circuit nominal et du circuit défectueux, alors que σ_n et σ_f représentent les écarts-types. De plus, peu importe l'algorithme de détection utilisé, les résultats de simulation ne servent qu'à déterminer si une panne est détectée ou non. Par conséquent, une faible précision des résultats est acceptable à condition que le critère de détection donne la même classification pour chaque panne.

1.3.2 Types de pannes

Deux types de pannes sont généralement considérés, les pannes paramétriques et catastrophiques. Tout d'abord, les pannes paramétriques résultent d'une déviation du comportement d'un composant due à une altération de sa géométrie lors de la fabrication. Par exemple, la valeur d'une résistance ou d'une largeur de transistor peut être altérée durant la fabrication, provoquant une modification du comportement du circuit. Ces variations sont surtout dues à un contrôle imparfait du procédé de lithographie [53]. De plus, elles peuvent être globales et affecter tous les composants d'un circuit intégré, ou locales et affecter quelques composants, ce qui peut causer des disparités (*mismatch*) entre ces composants [41]. Plusieurs valeurs de chaque paramètre influencé par le procédé de fabrication doivent être simulées afin de couvrir adéquatement les possibilités.

Les pannes dites catastrophiques causent des modifications structurelles sur le circuit. Ce sont principalement des courts-circuits (*short*) entre deux nœuds et des connexions ouvertes (*open*). Ces défauts peuvent être causés par la présence de poussières sur la surface du circuit ou d'un désalignement important des masques [41]. La liste de pannes catastrophiques à simuler peut être produite à partir du schéma du circuit en insérant des courts-circuits entre des terminaux de composants et en éliminant des connexions entre des nœuds du circuit [55]. Lorsqu'on connaît l'arrangement (*layout*) du circuit et le procédé de fabrication, des ajouts et manques de matériels peuvent être ajoutés de façon aléatoire sur des couches (*layer*) du circuit et le circuit défectueux résultant est extrait et analysé [20]. Cette méthode, appelée analyse inductive des pannes (*Inductive Fault Analysis ou IFA)*, est fortement recommandée mais peut nécessiter l'analyse d'un nombre très élevé de circuits défectueux [52].

1.3.3 Approches de simulation de pannes

Le temps de calcul requis par la simulation de pannes est très élevé, ce qui nécessite parfois la réduction du nombre de pannes à simuler pour conserver un temps de simulation acceptable. C'est pourquoi plusieurs méthodes ont été développées au cours des dernières années dans le but d'accélérer la simulation de pannes.

Simulation de pannes dans les circuits linéaires

Les circuits analogiques linéaires ont été abordés en premier lieu puisque, comme ils ne sont pas soumis à des commutations de composants non linéaires, ils ont un comportement beaucoup plus prévisible. Tout d'abord, [16] et [51] suggèrent d'exprimer la variation relative entre la sortie et chaque paramètre de composant à l'aide de la méthode de la matrice adjointe. La plus petite déviation détectable à la sortie peut alors être calculée pour chaque paramètre. Dans [63], chaque circuit défectueux est représenté par une matrice d'état ou encore par une perturbation de la matrice d'état du circuit nominal. La sortie est calculée en fonction des entrées, qui sont exprimées de façon polynomiale, et de la matrice d'état du circuit défectueux. Enfin, [35] propose de

représenter chaque panne par une équation supplémentaire dans le système MNA. La décomposition LU n'a pas à être répétée, ce qui réduit de façon importante le temps de simulation [3]. Toutefois, bien qu'elles puissent donner une bonne approximation pour les circuits faiblement non linéaires [67], ces méthodes sont difficilement applicables à l'injection de pannes dans les composants non linéaires tels les transistors.

Utilisation de modèles simplifiés

Une autre approche largement utilisée consiste à représenter les éléments du circuit à l'aide d'une description comportementale de haut niveau. Ainsi, beaucoup de temps de calcul est sauvé par rapport à l'évaluation des modèles très complexes de transistors. Tout d'abord, des blocs simples d'un circuit peuvent être représentés sous forme de macro-modèles comportementaux dans le *netlist*, c'est-à-dire à l'aide de source de tension / courant dépendantes / indépendantes et d'admittances. Les blocs dans lesquels une panne a été injectée sont alors décrits au niveau transistor [20] ou inclus dans les macro-modèles [69] et [43]. Des blocs complexes peuvent aussi être décrits par des modèles comportementaux faisant appel à des fonctions linéaires par partie (*piecewise linear* ou PWL) [34] et [8] ou à des courbes de régression (*regressive splines*) [11], mais un simulateur spécialisé est alors nécessaire. De plus, des langages de description de circuits mixtes analogique-numérique tels *VHDL-AMS* [24] et *Verilog-AMS* [1] ont fait leur apparition ces dernières années. Ils ont été utilisés en simulation de pannes dans [30] notamment. Toutefois, la modélisation précise de blocs demeure complexe, particulièrement pour les étages d'entrée et de sortie [13]. En fait, comme ces modélisations comportementales négligent plusieurs phénomènes de second ordre, elles peuvent difficilement fournir des résultats fiables pour toutes les topologies de circuit. De plus, elles doivent habituellement être générées par l'utilisateur, ce qui augmente le temps de préparation avant l'utilisation de telles méthodes sur un nouveau circuit.

Simulation de pannes en parallèle

Afin d'accélérer la simulation de pannes analogiques, il a été suggéré de simuler le circuit nominal et toutes les pannes simultanément, favorisant ainsi l'échange d'information d'un circuit à l'autre. Dans ce but, [68] réutilise les valeurs provenant de la linéarisation

des composants du circuit nominal pour les circuits défectueux qui possèdent des tensions de nœuds semblables à la même itération NR. Le temps nécessaire à l'évaluation des modèles de composants est ainsi grandement réduit pour les pannes qui affectent peu le circuit nominal. Pour leur part, [22] et [12] proposent des techniques pour accélérer la résolution du système matriciel dans le cas où la majorité des tensions de nœuds demeurent identiques entre le circuit nominal et chaque circuit défectueux. Toutefois, lorsqu'une panne affecte un nombre élevé de nœuds du circuit, ces méthodes perdent de leur efficacité.

Réduction du nombre d'itérations *Newton-Raphson*

La forme itérative de l'algorithme NR utilisé dans les simulateurs SPICE le rend très sensible à l'approximation initiale de la solution. Par conséquent, la simulation DC de pannes analogiques est accélérée de façon importante si une bonne approximation de la solution exacte (par exemple la solution du circuit nominal) est utilisée comme approximation initiale pour chaque circuit défectueux. Dans ce but, [58] propose d'ordonnancer les pannes de manière à ce que la solution du circuit défectueux précédant puisse fournir l'approximation initiale du circuit défectueux courant. Cet ordonnancement a été effectué à l'aide de la formule de *Householder* et nécessite une seule itération NR. Dans le cas de pannes paramétriques, [22] propose de calculer l'approximation initiale de chaque circuit défectueux à partir d'une interpolation linéaire des solutions obtenues pour les deux valeurs précédentes du paramètre défectueux. L'objectif de ces deux travaux consistait à obtenir la meilleure approximation possible de la solution de chaque circuit défectueux afin de diminuer le nombre total d'itérations à effectuer.

Lorsqu'on réutilise la solution du circuit nominal comme approximation initiale de la solution de chaque circuit défectueux, il a été démontré qu'une seule itération NR en utilisant la formule de *Householder* permet d'obtenir une bonne approximation du taux de couverture des pannes (*fault coverage*) [59]. D'ailleurs, lorsqu'on est en présence d'un circuit linéaire, une seule itération suffit pour obtenir la solution exacte. Cette méthode constitue en fait une linéarisation du circuit non linéaire autour de son point de

polarisation DC, alors que les pannes sont modélisées comme des pannes dans ce circuit linéarisé. L'utilisation d'un nombre fixe d'itérations a été expérimentée dans le cas de la génération de stimuli [65]. Plutôt que de simuler jusqu'à la convergence de l'algorithme NR, un nombre fixe d'itérations est effectué, puis la simulation est terminée et le stimulus présentant la meilleure couverture de pannes est sélectionné. Il a été montré que cette méthode présente le meilleur compromis précision / temps de calcul lorsque le nombre d'itérations est fixé à une seule itération plutôt que tout autre nombre supérieur. Toutefois, cette méthode ne permet aucun contrôle de la précision obtenue. Une extension de cette méthode a également été proposée pour la simulation transitoire [64].

1.3.4 Accélération de la simulation de pannes

Parmi les différentes méthodes de simulation de pannes présentées plus haut, nous avons choisi de nous concentrer sur celles qui permettent de diminuer le nombre d'itérations NR. D'une part, elles supportent les circuits analogiques non linéaires, ensuite elles se rapprochent de la simulation conventionnelle faisant appel aux simulateurs SPICE commerciaux et, enfin, elles ne nécessitent pas la modification des algorithmes d'évaluation des composants et de résolution du système matriciel présents dans les simulateurs SPICE. D'après les publications existantes telles [22] et [58], la réutilisation des résultats précédents afin de fournir une bonne approximation initiale de la solution pour chaque circuit défectueux contribue à réduire de façon importante le nombre d'itérations sans diminuer la précision. Or, si on n'effectue qu'une seule itération NR, il est possible d'obtenir des résultats fournissant une bonne approximation de la couverture de pannes [59] et [65]. Toutefois, aucun contrôle sur la précision n'est effectué, ce qui constitue un inconvénient majeur. Un compromis entre le nombre d'itérations et la précision de la simulation est étudié dans le chapitre 2 de ce mémoire. Le chapitre 4 présente une autre méthode permettant de faire un compromis précision / temps de calcul en fonction du seuil de détection de chaque panne. De plus, le chapitre 5 propose

notamment une méthodologie permettant de spécifier facilement l'approximation initiale de la solution de chaque circuit défectueux afin d'accélérer la simulation de pannes.

1.4 Dimensionnement automatique de circuits

Les outils de dimensionnement de circuits permettent de définir les dimensions optimales pour chaque composant d'un circuit afin d'atteindre les performances désirées. La topologie du circuit est généralement spécifiée par le concepteur, mais il existe quelques outils de synthèse automatique qui sélectionnent la topologie appropriée dans une bibliothèque [60] ou qui génèrent une topologie selon un algorithme évolutif [32]. Les outils dits « de synthèse » [25] et [33] génèrent les dimensions initiales du circuit, alors que les outils de dimensionnement automatique démarrent à partir d'un circuit initial proposé par le concepteur [45]. Dans les deux cas, des algorithmes permettent par la suite de déterminer les dimensions optimales des composants dans le but d'obtenir les performances désirées. Pour le reste de ce mémoire, nous ne ferons pas de distinction entre les algorithmes de synthèse et ceux de dimensionnement de circuits.

1.4.1 Historique du dimensionnement de circuits

Les premiers algorithmes de dimensionnement, « basés sur la connaissance » (*knowledge-based*), reposaient principalement sur des heuristiques définies par le concepteur selon la topologie du circuit [10] et [18]. Or, le développement d'équations à optimiser propres au circuit nécessite un certain temps de préparation de la part de l'utilisateur et ce, avant même de débuter le dimensionnement. Ces méthodes ont donc été remplacées par les approches « basées sur l'optimisation » (*optimization-based*), qui font appel à l'évaluation des performances de chaque circuit candidat dans la boucle interne d'un algorithme d'optimisation. Ces approches se divisent par la suite en deux sous-catégories. La première, l'« optimisation basée sur des équations » (*equation-based optimization*), utilise des équations analytiques afin d'évaluer les performances d'un

circuit candidat donné. Les équations peuvent être dérivées à la main [31], [19], ce qui peut être long et vulnérable à des erreurs humaines. Des techniques de simulation symbolique permettent de dériver automatiquement des équations simplifiées représentant le comportement du circuit [60]. Bien que ces équations permettent d'évaluer les performances du circuit dans un temps de simulation très court, plusieurs caractéristiques sont difficiles à capturer, notamment celles qui nécessitent une réponse transitoire ou grand-signal [15]. La seconde sous-catégorie, l'« optimisation basée sur la simulation » (*simulation-based optimization*), semble s'imposer alors que la disponibilité d'ordinateurs et d'algorithmes plus puissants rend une telle approche possible. Au lieu d'équations analytiques pour évaluer les performances d'un circuit candidat, une simulation complète est effectuée, ce qui permet de connaître ses performances exactes. Dans le cadre de ce mémoire, nous nous adresserons à ce type d'algorithme de dimensionnement de circuits.

1.4.2 Optimisation basée sur la simulation

Contrairement aux approches de dimensionnement de circuits énoncées au paragraphe précédent, l'optimisation basée sur la simulation profite de la précision des modèles de transistors encapsulés dans les simulateurs SPICE. De plus, leur flexibilité n'est pas limitée puisque les simulateurs commerciaux supportent tous les types de circuit. Enfin, comme le concepteur n'a pas à fournir d'équations modélisant le comportement du circuit, le temps de préparation requis pour le dimensionnement d'un circuit donné se limite à la spécification des objectifs et contraintes. Le schéma général de fonctionnement est illustré à la Figure 1.2.

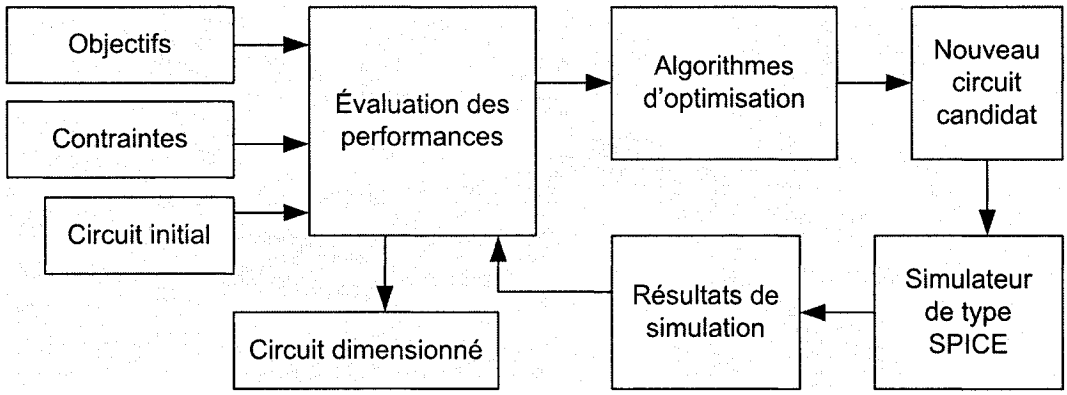


Figure 1.2 : Schéma-bloc du dimensionnement basé sur la simulation

Tout d'abord, le *netlist* du circuit à dimensionner doit être fourni. De plus, une liste de spécifications à rencontrer (objectifs) est aussi fournie, accompagnée d'une liste de contraintes dures et de contraintes douces. Les premières doivent être absolument rencontrées alors que les secondes peuvent ne pas être respectées en totalité dans le circuit final. Les performances de chaque circuit candidat sont évaluées à partir de ses résultats de simulation, puis l'algorithme d'optimisation propose, à partir de ses degrés de liberté, un nouveau dimensionnement prometteur du circuit. Lorsqu'un candidat atteint les objectifs et respecte toutes les contraintes, il est retourné en tant que circuit optimal. Le nombre de candidats explorés peut dépasser 100 000 [48].

1.4.3 Approches de dimensionnement basé sur la simulation

Cette sous-section décrit quelques outils de dimensionnement ou de synthèse de circuits proposés dans la littérature. L'emphase sera mise sur l'accélération de la simulation plutôt que sur les algorithmes de dimensionnement eux-mêmes. Tout d'abord, l'outil *DELIGHT.SPICE* [45] utilise les fonctions de calcul des sensibilités (*sensitivity*) disponibles dans le simulateur *SPICE3* [49] afin d'optimiser le nombre de simulations. Or, comme dans le cas de l'outil d'optimisation disponible dans *HSPICE* [4], seule une solution locale, près du dimensionnement initial du circuit, peut être espérée. Dans l'outil présenté dans [38], des fonctions permettant la réutilisation des résultats de simulation

des circuits précédents afin d'accélérer la simulation DC ont été ajoutées dans *SPICE3*. Un simulateur adapté à l'outil de dimensionnement de circuits permettant la réutilisation des solutions DC est également utilisé dans [2]. Afin de diminuer le temps de calcul, *MAELSTROM* [33] et *ANACONDA* [48] distribuent les simulations en parallèle sur plusieurs ordinateurs. Contrairement à [38], cette dernière approche ainsi que *AMIGO* [25] ont l'avantage d'utiliser un simulateur commercial encapsulé, ce qui préserve l'indépendance entre l'outil de dimensionnement et le simulateur. Il a aussi été proposé de ne simuler qu'une fraction des circuits candidats, les performances des autres pouvant être évaluées à l'aide d'approximations, ce qui réduit le temps de simulation global [17]. Selon [15], des recherches permettant une diminution importante du temps de calcul sont nécessaires avant de voir des outils de dimensionnement automatique de circuits utilisés de façon courante par les concepteurs de circuits analogiques.

1.4.4 Accélération de la simulation pour le dimensionnement de circuits

Les travaux exposés plus haut peuvent se diviser en deux catégories : ceux qui utilisent leur propre simulateur [2] et [38], et ceux qui utilisent un simulateur commercial [33], [48] et [17]. Les premiers sont potentiellement plus rapides que les seconds, mais nécessitent le développement de leur propre simulateur. De même, les simulateurs développés pour une application donnée sont difficilement réutilisables par d'autres applications. Une caractéristique importante de ces simulateurs dédiés est le partage de l'information d'un circuit à l'autre, particulièrement en ce qui a trait à la solution DC. De plus, une réduction de la précision contre une accélération de la simulation est possible, ce qui n'est pas le cas avec les simulateurs commerciaux, sauf en éliminant certaines simulations comme dans [17]. Un simulateur général favorisant la réutilisation des résultats de simulation précédents pour accélérer les simulations à venir tout en permettant d'ajuster la précision selon les besoins serait un excellent compromis. Le chapitre 5 propose plusieurs fonctionnalités permettant d'obtenir un tel compromis au

niveau du simulateur afin de réduire de façon significative la durée des simulations, sans pour autant rendre le simulateur dédié à une application donnée.

1.5 Analyse Monte Carlo

L'analyse Monte Carlo consiste à soumettre un circuit analogique à des variations aléatoires qui représentent les effets des fluctuations du procédé de fabrication du circuit. L'objectif est de connaître l'influence de ces variations de paramètres sur le fonctionnement du circuit. Comme nous le verrons plus loin, l'analyse Monte Carlo n'est pas une application en soit, mais est habituellement utilisée afin d'estimer et d'optimiser la robustesse du circuit ainsi que le rendement à la fabrication. Elle est également de plus en plus utilisée par les applications de simulation de pannes.

1.5.1 Fonctionnement général

L'analyse Monte Carlo est disponible dans plusieurs simulateurs commerciaux tels *HSPICE* et *Spectre*. Le schéma général est le suivant :

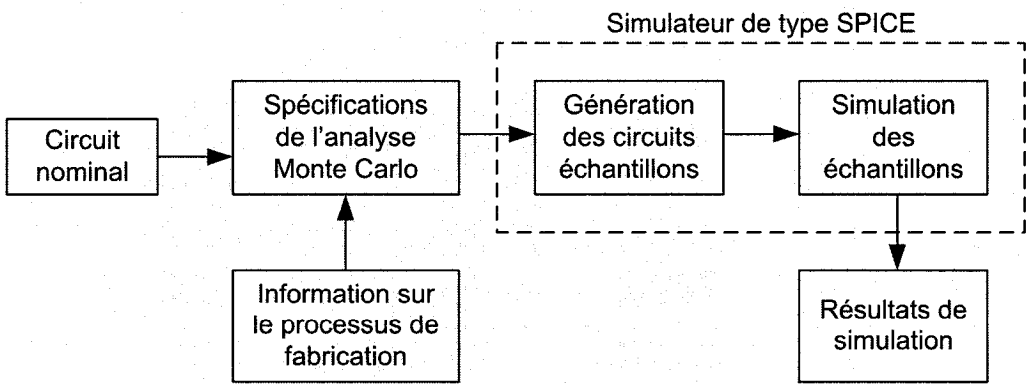


Figure 1.3 : Schéma-bloc de l'analyse Monte Carlo dans un simulateur

Tout d'abord, les spécifications de l'analyse Monte Carlo doivent être définies à partir du circuit nominal et des informations sur le procédé de fabrication. L'analyse Monte Carlo est par la suite appelée dans le simulateur en spécifiant un nombre d'échantillons à simuler. Le simulateur génère lui-même les échantillons de circuit correspondant au procédé de fabrication. Une fois chacun des circuits échantillons simulés, les résultats de simulation sont retournés. L'inconvénient principal d'utiliser la fonction Monte Carlo d'un simulateur commercial est l'absence de contrôle sur les échantillons générés ainsi que sur la précision de la simulation. De plus, l'utilisateur est limité aux fonctions offertes par le simulateur pour définir les variations aléatoires des paramètres.

Deux types de variations de paramètres sont généralement considérés : les variations locales (*intra-die*) et les variations globales (*inter-die*). Les premières, aussi appelées disparités entre composants (*mismatch*), sont définies comme la différence entre les caractéristiques de composants destinés à se comporter de façon identique ou symétrique sous les mêmes conditions de polarisation. Les secondes sont plutôt des variations identiques pour tous les composants d'un type donné sur une puce ou un lot donné. Ces deux types de variations de paramètres sont souvent étudiés en l'analyse Monte Carlo.

1.5.2 Applications ayant recours à l'analyse Monte Carlo

L'étude de la robustesse d'un circuit analogique permet de connaître son fonctionnement dans les pires conditions [54]. La robustesse d'un circuit a également une grande influence sur le rendement de fabrication. Le rendement (*yield*) d'un circuit est défini comme le pourcentage de puces produites qui rencontrent les spécifications par rapport au nombre total de puces fabriquées. L'analyse Monte Carlo est utilisée pour évaluer de quelle manière les caractéristiques de ces puces varieront d'un échantillon à l'autre avant de les fabriquer. On recherche la proportion d'écantillons qui, une fois simulés, rencontrent les spécifications du circuit [46]. Dans le cas de l'optimisation du rendement, on recherche une conception du circuit nominal pour laquelle la proportion d'échantillons

acceptés est la plus élevée. L'analyse Monte Carlo se retrouve donc à l'intérieur d'une boucle d'optimisation, comme présenté dans [57].

Les méthodes récentes de simulation de pannes considèrent les variations aléatoires des paramètres des composants. À cette fin, l'équation 1.6 de la section 1.3.1 représente le seuil de détection pour chaque panne. La distribution de la tension de sortie pour le circuit nominal et chaque circuit défectueux est calculée à partir des résultats fournis par l'analyse Monte Carlo. Plus le nombre d'échantillons est élevé, plus les résultats sont représentatifs, mais le temps de calcul augmente proportionnellement. Dans [47], l'analyse Monte Carlo est effectuée pour le circuit nominal et chaque circuit défectueux, ce qui nécessite un temps de calcul élevé. L'analyse Monte Carlo peut être effectuée seulement sur les pannes difficilement détectables [56]. De même, [28] suggère de diminuer le nombre d'échantillons pour les pannes plus facilement détectables. Toutefois, le nombre d'échantillons à simuler peut demeurer élevé pour un grand nombre de pannes, ce qui requiert tout de même un temps de calcul élevé.

1.5.3 Accélération de la simulation pour l'analyse Monte Carlo

La ressemblance entre chaque circuit nominal et les échantillons étant généralement grande, des méthodes ont été développées afin d'accélérer l'analyse Monte Carlo. Ainsi, [66] propose une approximation linéaire en deux étapes pour chaque échantillon. Une seule itération NR est alors effectuée à partir de la solution du circuit nominal. De plus, la coûteuse factorisation LU est évitée par la réutilisation du jacobien du circuit nominal et l'utilisation d'un vecteur de correction. Une autre méthode suggérée dans [36] évite aussi la factorisation LU pour la première itération, mais effectue des itérations jusqu'à la convergence si plus d'une itération sont nécessaires. Toutefois, cette méthode n'est efficace que si le nombre d'échantillons est beaucoup plus élevé que le nombre de paramètres modifiés, puisque le calcul de la sensibilité de la sortie par rapport à chaque paramètre aléatoire est nécessaire. Ces deux méthodes perdent leur efficacité lorsque

plusieurs itérations NR sont nécessaires, puisque la première perd de la précision alors que la seconde ne réduit plus le temps de calcul jusqu'à la convergence. Il est donc souhaitable de réduire le temps de calcul tout en assurant une précision acceptable lorsque plusieurs itérations sont requises avant la convergence. Un tel compromis entre le nombre d'itérations et la précision de la simulation est présenté au chapitre 2.

1.6 Approches générales d'accélération de la simulation

Pour les trois applications visées par ce projet de recherche, nous avons jusqu'ici mis l'accent sur les méthodes qui permettent de réduire le nombre d'itérations NR nécessaires pour effectuer les simulations DC. Nous pouvons classer ces méthodes en deux classes principales. La première consiste à favoriser le partage d'information d'un circuit modifié à l'autre, alors que la seconde permet une diminution de la précision contre une diminution du temps de calcul.

La réutilisation de l'information d'un circuit à l'autre permet d'éviter de relire le circuit au complet à chaque fois. De plus, il est possible de débuter chaque simulation DC par une approximation initiale qui risque d'être proche de la solution finale, ce qui réduit le nombre d'itérations NR. Pour chacune des applications, une approximation initiale précise de la solution (AIPS) peut être utilisée pour chaque circuit modifié. Dans le cas du dimensionnement de circuits, cette approximation provient de circuits précédemment simulés. Pour la simulation de pannes, elle est fournie soit par le circuit nominal ou par des circuits défectueux précédents. Enfin, pour l'analyse Monte Carlo elle provient du circuit nominal. Les méthodes proposées aux chapitres 2 et 4 nécessitent toutes deux une AIPS. De plus, le simulateur proposé au chapitre 5 permet de spécifier une AIPS pour chaque circuit modifié.

L'utilisation d'une AIPS ouvre également la porte à un compromis entre le temps de calcul et la précision de la simulation. En effet, tant en analyse Monte Carlo qu'en

simulation de pannes, il a été suggéré de n'effectuer qu'une itération NR lorsqu'on démarre d'une approximation initiale appropriée. Toutefois, un meilleur contrôle de la précision des résultats est souhaitable afin d'assurer le bon fonctionnement des applications visées. C'est pourquoi deux méthodes permettant un compromis entre le nombre d'itérations et la précision des résultats obtenus sont proposées dans les chapitres 2 et 4 de ce mémoire.

1.7 Conclusion

Dans ce chapitre, nous avons vu les notions de base de la simulation de circuits analogiques. Ensuite, nous avons fait un survol des applications visées dans le présent mémoire, à savoir la simulation de pannes, le dimensionnement automatique de circuits, et l'analyse Monte Carlo. Nous avons fait une revue des méthodes retrouvées dans la littérature qui permettent d'accélérer la simulation pour chacune de ces applications. Nous nous sommes particulièrement intéressés aux méthodes d'accélération de la simulation DC faisant appel à un simulateur de type SPICE.

Nous avons constaté que l'utilisation d'une AIPS pour chaque circuit modifié est une méthode éprouvée pour diminuer le nombre d'itérations NR nécessaires à l'obtention des résultats de simulation. Les applications visées sont toutes en mesure de fournir une telle AIPS à condition de conserver les résultats obtenus des simulations précédentes. De même, nous avons vu qu'il est possible de terminer les itérations NR avant la convergence tout en obtenant des résultats relativement significatifs, tant en simulation de pannes qu'en analyse Monte Carlo. Le temps de simulation est alors grandement réduit. Même si cela n'a jamais été expérimenté à notre connaissance, l'arrêt des itérations avant la convergence est également envisageable pour le dimensionnement de circuits.

Le prochain chapitre fait l'analyse de l'algorithme itératif NR et propose une méthode permettant d'arrêter les itérations avant la convergence lorsque la précision désirée est

atteinte. Pour sa part, le chapitre 4 propose un nouveau critère d'arrêt des itérations avant la convergence s'appliquant bien à la simulation de pannes et qui se base sur le seuil de détection de chaque panne. Enfin, le chapitre 5 présente un simulateur SPICE optimisé pour les applications nécessitant des simulations multiples. Des applications de simulation de pannes, de dimensionnement de circuits et d'analyse Monte Carlo ont été développées afin d'exploiter ce simulateur et y sont présentées.

CHAPITRE 2

Augmentation de la tolérance relative

2.1 Introduction

Dans ce chapitre, l'algorithme itératif NR est étudié un peu plus en profondeur dans le cas de la simulation DC d'un circuit analogique. L'objectif est de réduire le nombre d'itérations qui doivent être effectuées afin d'obtenir des résultats avec une précision suffisante pour les besoins de l'application visée. En d'autres mots, il s'agit d'obtenir un compromis entre le temps de calcul et la précision des résultats obtenus.

Dans un premier temps, des exemples illustrant la convergence d'un circuit analogique en analyse DC sont présentés. L'importance de l'approximation initiale utilisée ainsi que la proximité de la solution exacte lors des dernières itérations avant la convergence sont mises en évidence. Une métrique est ensuite proposée afin d'évaluer à quel point la solution approximative à l'itération actuelle se rapproche de la solution exacte obtenue à la convergence.

Enfin, il est montré expérimentalement que, dans le cas où l'algorithme NR est démarré à partir d'une AIPS, l'utilisation d'une valeur de tolérance relative plus grande permet de terminer l'algorithme NR plus rapidement tout en obtenant une erreur inférieure à la tolérance utilisée.

2.2 Convergence en simulation DC

2.2.1 Utilisation d'une approximation initiale précise de la solution

Tel que mentionné à la section 1.2, les simulateurs analogiques font appel à l'algorithme NR pour résoudre le système d'équations non linaires qui décrit un circuit donné. L'algorithme démarre à partir d'une approximation initiale quelconque de la solution et se termine après un certain nombre d'itérations lorsqu'il converge vers la solution exacte. La Figure 2.1 (a) illustre un cas typique de convergence DC lorsque aucune approximation initiale précise n'est disponible. La simulation DC démarre donc à partir d'une solution où toutes les tensions sont nulles. La tension de sortie oscille durant quelques itérations, puis converge vers sa valeur finale. Les 7 sept premières itérations peuvent être considérées comme du temps de calcul perdu puisque l'algorithme n'est pas en train de converger. Toutefois, à l'itération 8, une valeur près de la valeur exacte est obtenue. La convergence est donc atteinte après quelques itérations additionnelles.

Pour sa part, la Figure 2.1 (b) illustre le comportement du même circuit, profitant cette fois d'une AIPS fournie par un circuit similaire avec de légères différences. La convergence est atteinte en quatre itérations seulement, sans oscillations de tension notables. Le système d'équations non linéaire qui décrit le circuit demeure stable et converge rapidement. La réutilisation de la solution d'un circuit similaire réduit le nombre d'itérations et accélère la convergence en fournissant immédiatement une solution approximative proche de la solution exacte, où la convergence est pratiquement quadratique tel que mentionné à la section 1.2.

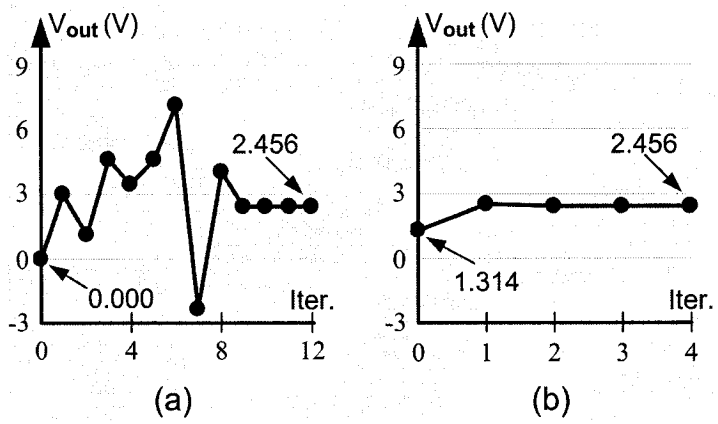


Figure 2.1 : Tension de sortie durant les itérations NR : (a) approximation initiale quelconque; (b) approximation initiale précise.

2.2.2 Variation relative maximale

La précision d'une solution intermédiaire ne peut être connue de façon précise que lorsque la solution exacte est connue. Or, il est souhaitable d'obtenir un indicateur de la précision après chaque itération sans devoir effectuer d'itérations supplémentaires. Par conséquent, une équation devait être déterminée afin d'approximer la précision atteinte après chaque itération et ce, à partir des résultats observables à cette même itération. Divers paramètres ont donc été observés après chaque itération, notamment :

- variation relative de tension à chaque nœud,
- variation absolue de tension à chaque nœud,
- modification des valeurs de conductance dans la matrice MNA,
- détection d'une commutation de transistor entre deux itérations,
- et proportion des nœuds dont la tension a varié significativement.

Des résultats expérimentaux ont permis de déterminer que la variation relative de tension à chaque nœud est la mesure la plus fiable de la précision d'un circuit. Une combinaison des divers paramètres ne permettait pas d'améliorer les résultats donnés par la variation relative seule. Une métrique tenant compte uniquement de la variation relative à chaque

nœud a alors été trouvée. Cette nouvelle métrique, appelée *Variation Relative Maximale* (*VarRelMax*), représente la plus grande variation relative affectant les nœuds du circuit entre deux itérations consécutives k-1 et k et est exprimée par l'équation 2.1.

$$\boldsymbol{VarRelMax} = \max_{\forall i}\left(\frac{|\, x_i^k - x_i^{k-1}\,|}{\max\left(|\, x_i^k\,|, |\, x_i^{k-1}\,|, \boldsymbol{abstol}\right)}\right) \qquad (2.1)$$

Dans cette équation, x_i^k est le résultat intermédiaire obtenu pour le nœud i après l'itération k et *abstol* est le paramètre de tolérance absolue abordé à la section 1.2. À chaque itération, on mesure donc la variation relative obtenue pour chacun des nœuds du circuit, puis on conserve la valeur de variation relative la plus élevée parmi tous les nœuds. Ainsi, on ne tient pas seulement compte des nœuds de sortie, mais bien de l'ensemble des nœuds du circuit. L'idée est d'interpréter $VarRelMax^k$ comme la précision atteinte par la solution intermédiaire x^k, de telle sorte que l'erreur de la tension de sortie est considérée comme étant toujours inférieure à cette métrique. Cette assertion est valide dans le cas de la convergence quadratique, puisque l'erreur diminue par plus d'un ordre de grandeur entre deux itérations consécutives.

Pour l'exemple de la section précédente (Figure 2.1), la Figure 2.2 (a) montre l'erreur d'approximation de la tension de sortie à chaque itération, qui diminue rapidement et montre une convergence quasi-quadratique. La Figure 2.2 (b) illustre la métrique de variation relative maximale pour le même exemple. La valeur de $VarRelMax^k$ diminue rapidement mais demeure en tout temps supérieure à l'erreur de la tension de sortie à chaque itération. La variation relative maximale constitue dans cet exemple une borne supérieure de l'erreur commise à n'importe lequel des nœuds du circuit. Comme nous le verrons plus loin, $VarRelMax^k$ ne borne pas correctement l'erreur dans tous les cas et son utilisation comme approximation de l'erreur demeure donc une heuristique.

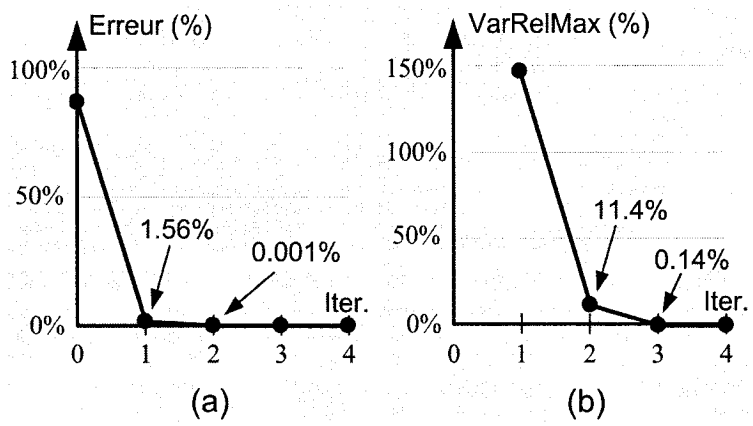


Figure 2.2 : Itérations NR dans un cas de convergence rapide : (a) erreur d'approximation de la sortie; (b) variation relative maximale

On remarque que le nombre d'itérations peut être réduit si l'algorithme NR est arrêté dès qu'une précision suffisante est atteinte. En effet, dans l'exemple de la Figure 2.1 (b) et de la Figure 2.2 (a), la simulation peut être terminée après une seule itération si une précision de ±2 % est suffisante, alors qu'une deuxième itération réduit l'erreur à une valeur négligeable. De son côté, *VarRelMax* indique que l'erreur est inférieure à 11.4 % après deux itérations, tel qu'illustré à la Figure 2.2 (b). Par conséquent, si une erreur de 12 % est jugée acceptable, le nombre d'itérations peut être réduit de quatre à deux sans affecter les résultats désirés puisque l'erreur est, en pratique, inférieure à *VarRelMax*. Si une précision de ±2 % est nécessaire, une troisième itération doit être effectuée, ce qui donne tout de même un gain en temps de calcul de 25 %.

2.2.3 Convergence lente

Dans l'exemple précédent, la convergence est rapide lorsqu'on démarre l'algorithme NR à partir d'une AIPS. Par contre, si cette approximation initiale est éloignée de la solution exacte, des oscillations importantes peuvent survenir avant la convergence. La Figure 2.3 illustre un tel exemple. La tension de sortie effectue une oscillation à l'extérieur de l'intervalle de tensions possibles à l'itération 4, puis finit par se stabiliser après un certain

nombre d'itérations additionnelles, tel qu'illustré à la Figure 2.3 (a). La métrique *VarRelMax* diminue tout d'abord, puis augmente avant de diminuer jusqu'à une valeur proche de zéro. Selon la Figure 2.3 (b) et (c), si une précision de ±2 % est requise, neuf itérations sont effectuées pour une réduction de 18 % et une erreur sur la sortie de 0.3 %. Toutefois, si une erreur de 7 % est acceptable, le nombre d'itérations peut être réduit de 27 %, ce qui donne une erreur de 1 %. Dans ce même exemple, on remarque également qu'il est impossible d'effectuer moins de cinq itérations et obtenir une solution se rapprochant de la solution exacte. Par le fait même, effectuer une seule itération, tel que proposé dans [59] et [65], produit à coup sûr des résultats erronés.

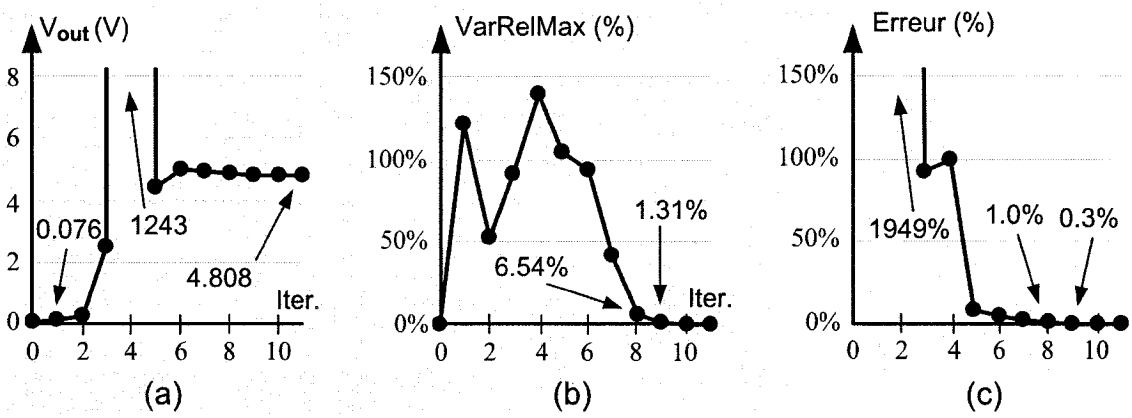


Figure 2.3 : Itérations NR dans un cas de convergence lente : (a) tension de sortie; (b) variation relative maximale; (c) erreur d'approximation de la tension sortie

2.3 Méthode d'augmentation de la tolérance relative

Les exemples de la section précédente amènent à proposer une méthode d'arrêt des itérations avant la convergence en simulation DC. On retrouve une corrélation évidente entre le critère de convergence présenté aux équations (1.2) et (1.3) et *VarRelMax* : si, à une itération k, $VarRelMax^k$ est inférieure au paramètre *reltol*, le critère de convergence termine les itérations. Il est donc possible de réutiliser le critère habituel de convergence comme un critère d'arrêt des itérations avant convergence.

L'algorithme complet de la méthode d'augmentation de la tolérance relative proposée est présenté à la Figure 2.4. Tout d'abord, une AIPS est définie à partir de la solution obtenue pour un circuit semblable à celui qui est simulé. Par la suite, le paramètre de tolérance relative *reltol* est considéré comme une mesure de la précision à atteindre, c'est-à-dire qu'on s'attend à ce que l'erreur commise sur la solution générée soit inférieure à la tolérance relative fixée. Ainsi, les itérations sont terminées dès que la variation relative maximale est inférieure à la valeur donnée au paramètre de tolérance relative *reltol*. La simulation s'arrête donc une fois que la précision requise est atteinte plutôt qu'à la solution exacte à la convergence. Puisque les dernières itérations sont évitées, une réduction du nombre total d'itérations NR est obtenue. On obtient alors un compromis entre le nombre d'itérations effectuées et la précision des résultats obtenus.

1 *reltol* ← précision requise pour la simulation
2 Définir le stimulus DC à appliquer
3 Définir une AIPS x^0
4 k ← 0
5 arrêt ← faux
6 **Répéter**
7 k ← k + 1
8 Effectuer l'itération k pour obtenir x^k
9 $$VarRelMax^k = \max_{\forall i}\left(\frac{|x_i^k - x_i^{k-1}|}{\max(|x_i^k|, |x_i^{k-1}|, abstol)}\right)$$
10 **Si**($VarRelMax^k < reltol$)
11 arrêt ← vrai
12 **Fin Si**
13 **Tant que** (arrêt = faux)

Figure 2.4 : Algorithme de la méthode d'augmentation de la tolérance relative

Par définition, la convergence se produit lorsque la solution exacte est atteinte, tel que mentionné à la section 1.2. En pratique toutefois, une très faible tolérance est acceptée par les simulateurs, généralement 0.1 % ou moins. Pour une telle valeur de tolérance,

l'erreur commise sur les résultats obtenus est inférieure à la précision des modèles de transistors utilisés. Les résultats obtenus sont donc assumés exacts et la convergence est considérée comme atteinte. Dans la méthode d'augmentation de la tolérance relative proposée, cette tolérance est augmentée de façon très importante. Par exemple, si *reltol* est augmenté à une précision désirée de 5 %, les résultats obtenus peuvent différer de 5 % des résultats exacts, ce qui peut être largement supérieur à l'imprécision des modèles de transistors. En d'autres mots, en utilisant une telle valeur de tolérance relative, on ne peut assumer que la convergence est atteinte lorsque les itérations se terminent. Les résultats obtenus sont alors considérés comme approximatif, mais la précision attendue est égale à la valeur donnée à *reltol*. Par conséquent, bien que la méthode proposée utilise le critère habituel d'arrêt des itérations, les résultats obtenus ne sont pas exacts, mais sont une approximation plus ou moins précise des résultats exacts.

Puisque le nombre d'itérations requises dans les cas de convergence lente peut être beaucoup plus élevé que dans l'exemple précédent, le gain en terme de nombre d'itérations est donc plus faible que dans les cas de convergence rapide. De plus, si le nombre d'itérations est trop élevé et que la convergence n'est pas atteinte, les simulateurs SPICE utilisent des méthodes de continuation [39] pour trouver une solution à l'aide d'étapes intermédiaires. La réduction du nombre d'itérations devient alors donc négligeable puisque plusieurs centaines d'itérations sont effectuées. La proportion de cas de convergence rapide affecte par conséquent le gain potentiel obtenu par la méthode d'augmentation de la tolérance relative.

Lorsque la valeur de tolérance relative est augmentée, tel que proposé dans ce chapitre, la validité des résultats ne peut être assumée que dans les cas de convergence rapide. En effet, puisque l'erreur décroît très rapidement, *VarRelMax* surestime l'erreur, tel qu'illustré dans la Figure 2.2. Par contre, dans les cas de convergence lente, l'erreur peut osciller et *VarRelMax* ne surestime pas toujours l'erreur commise. Par conséquent, il y a un risque d'obtenir des résultats erronés si les itérations sont terminées prématurément.

Les résultats expérimentaux présentés dans la section suivante permettent de mieux quantifier ce risque.

2.4 Résultats expérimentaux

Le simulateur *SPICE3f5* [49] a été modifié et utilisé afin de valider cette méthode d'arrêt des itérations avant la convergence. Les modèles de transistors ainsi que l'algorithme utilisé pour effectuer une itération NR ont été conservés intacts. Les techniques de continuation offertes par *SPICE3f5* sont utilisées si la convergence n'est pas atteinte après 100 itérations. Les circuits analogiques de référence utilisés proviennent de ITC'97 [26] ainsi que de CircuitSim90 [37].

Les résultats expérimentaux sont présentés dans le Tableau 2.1. La première colonne donne le nom de chaque circuit de référence alors que la seconde colonne spécifie la variation des paramètres du circuit. Pour chaque circuit de référence, des circuits modifiés ont été générés à l'aide d'une variation aléatoire des paramètres de type Monte Carlo et selon une distribution uniforme. Pour ce faire, chaque résistance, largeur et longueur de grille de transistor MOS et facteur de surface de transistor BJT a été soumis à une variation aléatoire. Des variations de 10 % (petites), puis 50 % (grandes) des valeurs nominales de ces paramètres ont été expérimentées, tel que présenté à la deuxième colonne. Dans chaque cas, un échantillon de 500 circuits modifiés a été généré. La solution du circuit de référence a été réutilisée comme AIPS pour chaque circuit modifié. La troisième colonne donne le pourcentage de circuits modifiés qui ont présenté une convergence lente, alors que la quatrième indique le pourcentage de circuits modifiés qui ont fait appel aux techniques de continuation pour l'obtention d'une solution. Les colonnes 5, 6 et 7 présentent le nombre moyen d'itérations requises par circuits modifié et ce, pour des valeurs de *reltol* de 0.1 % (convergence), 5 % (précision moyenne) et 20 % (faible précision). Le gain en itérations obtenu pour les deux précisions désirées par rapport à la simulation jusqu'à la convergence est montré dans les colonnes 8 et 9. Enfin,

les colonnes 10 et 11 rapportent le pourcentage de circuits modifiés (sur les 500) pour lesquels la tension de sortie obtenue est erronée. Un tel résultat erroné survient lorsque la tension de sortie obtenue diffère de sa valeur exacte par plus que l'erreur acceptable donnée par *reltol*.

Tableau 2.1 : Résultats expérimentaux pour la méthode d'augmentation de la tolérance relative sur 500 essais

Circuit	Variation des paramètres (%)	Type de convergence (%)		Nombre moyen d'itérations			Gain en itérations (%)		Solutions obtenues erronées (%)	
		Convergence lente	Méthodes de continuation	Convergence	*Reltol*					
					5%	20%	5%	20%	5%	20%
leapfrog	10	1.6	0.0	3.12	2.55	2.16	18.3	30.8	0.0	0.0
	50	25.4	0.0	4.49	3.87	3.37	13.8	24.9	0.0	0.0
statevar	10	1.4	0.0	3.17	2.53	2.15	20.2	32.2	0.0	0.0
	50	7.4	0.8	5.49	4.49	4.06	18.2	26.0	0.0	0.0
ab_integ	10	0.0	0.0	3.06	2.03	1.95	33.7	36.3	0.0	0.0
	50	18.4	0.6	6.10	4.69	3.94	23.1	35.4	0.6	1.2
bias	10	0.0	0.0	2.34	1.00	1.00	57.3	57.3	0.0	0.0
	50	22.6	0.0	4.24	3.40	1.49	19.8	64.9	0.2	0.2
vreg	10	0.0	0.0	2.00	1.16	1.00	42.0	50.0	0.0	0.0
	50	0.0	0.0	2.59	1.93	1.24	25.5	52.1	0.0	0.0
gm6	10	10.2	0.0	4.43	3.09	2.51	30.2	43.3	1.4	1.8
	50	36.0	0.0	5.89	3.96	2.92	32.8	50.4	15.6	30.0
Moyenne	10	2.2	0.0	3.02	2.06	1.79	33.6	41.6	0.2	0.3
	50	18.3	0.2	4.80	3.72	2.84	22.2	42.3	2.7	5.2

On remarque tout d'abord qu'une grande variation des valeurs des paramètres fait en sorte que l'approximation initiale utilisée s'éloigne de la solution qui sera obtenue après simulation. Les échantillons diffèrent alors de façon importante du circuit initial. Par conséquent, le pourcentage d'échantillons qui présentent une convergence lente est beaucoup plus élevé que dans le cas d'une petite variation des paramètres. Dans 0.2 % des cas, plus de 100 itérations sont nécessaires, nécessitant des techniques de continuation pour trouver une solution. Pour la même raison, le nombre moyen d'itérations est plus élevé pour de grandes variations, c'est-à-dire 4.80 versus 3.02

itérations. Si une précision de 5 % est requise, la méthode proposée permet d'obtenir une réduction du nombre d'itérations de 33.6 % pour de grandes variations de paramètres et de 22.2 % pour de petites variation. Une précision de 20 % permet pour sa part une réduction de l'ordre de 40 % du nombre d'itérations.

En ce qui concerne la précision des résultats obtenus, on remarque que le nombre de résultats erronés est plus faible lorsque les circuits de référence sont soumis à des petites variations. Ceci est dû au fait que, dans ce cas, l'approximation initiale utilisée est beaucoup plus proche de la solution finale. Par conséquent, la proportion de cas de convergence lente, susceptibles de causer des erreurs, est largement réduite. De plus, le nombre de résultats erronés est plus faible lorsque la précision désirée est plus élevée. Ce résultat s'explique par le fait que *reltol* se rapproche alors de la valeur pour laquelle la convergence est assumée, c'est-à-dire 0.1 %. Le risque que les itérations soient terminées prématurément s'en trouve donc réduit. Globalement, la méthode proposée ne produit que 0.2 à 0.3 % de résultats erronés en moyenne dans le cas de circuits légèrement modifiés. Si les circuits sont altérés de façon importante, le risque de résultats erronés est de 2.7 à 5.3 %, dépendant de la précision requise.

Parmi les circuits de référence utilisés, *gm6* produit à lui seul la majorité des résultats erronés. Il s'agit d'un circuit amplificateur en boucle ouverte, dont la sortie est très sensible aux variations, ce qui fait en sorte que l'approximation initiale de la solution est souvent imprécise. La méthode proposée est donc moins efficace sur ce circuit particulier que sur les cinq autres, qui sont moins sensibles aux variations.

La Figure 2.5 illustre un cas de convergence susceptible de produire des résultats erronés avec la méthode d'augmentation de la tolérance relative. Tel qu'illustré en (a), le circuit semble converger vers une tension de sortie autour de -0.80 V, alors que la valeur exacte est plutôt de 0.698 V. En effet, à la cinquième itération, aucun nœud n'a subi une variation de tension supérieure à 4.76 %, tel qu'illustré en (b). La méthode proposée

termine donc les itérations à ce moment si *reltol* est fixé à 5 %, ce qui génère des résultats erronés. Dans un tel cas, seule une valeur faible de *reltol* est susceptible de produire les résultats attendus.

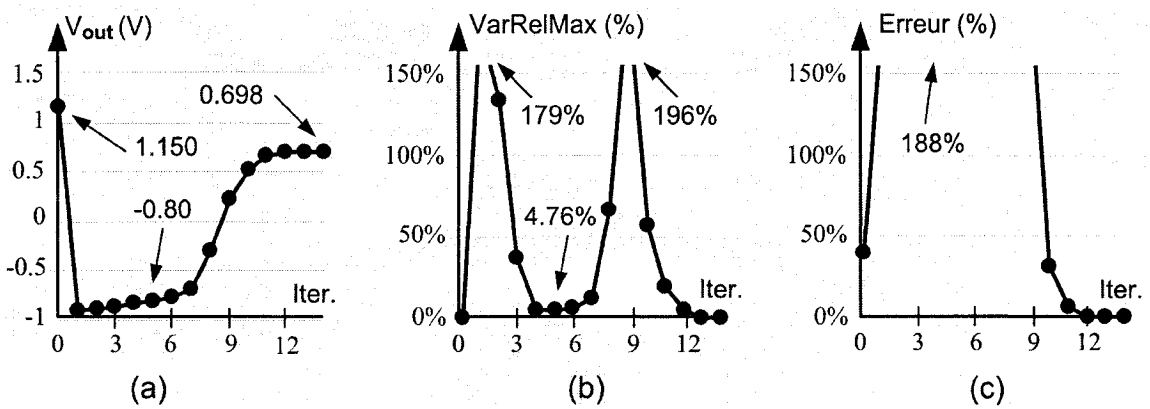


Figure 2.5 : Itérations NR dans un cas produisant des résultats erronés : (a) tension de sortie; (b) variation relative maximale; (c) erreur d'approximation à la sortie

2.5 Conclusion

Ce chapitre proposait une méthode permettant de réduire la précision de la simulation DC afin de diminuer le temps de calcul requis pour produire les résultats de simulation. Cette méthode propose de démarrer la simulation à partir d'une AIPS fournie par un circuit semblable dont on connaît les résultats de simulation. Le paramètre de tolérance relative, *reltol*, est alors fixé à la précision requise par les résultats de simulation. L'algorithme NR se termine par conséquent avant d'atteindre la convergence, mais les résultats approximatifs générés sont considérés comme suffisamment précis pour les besoins de l'application visée.

Cette méthode a été expérimentée sur un ensemble de circuits de référence pour différentes modifications et précisions requises. Dans chaque cas, le circuit original fournit l'AIPS pour chacun des circuits modifiés. Le gain en terme d'itérations NR ainsi

que la validité des résultats obtenus dépendent principalement des modifications au comportement du circuit initial ainsi que de la précision requise :

1) *Altération du comportement du circuit* : Si les modifications affectant un circuit de référence sont faibles, leur effet sur le comportement du circuit favorise l'utilisation d'une approximation initiale plus proche de la solution exacte, puisque celle-ci est plus facilement prévisible. La proportion de cas de convergence lente et, par conséquent, le risque d'obtention de résultats erronés, est réduit.

2) *Précision requise* : Une faible précision requise (*reltol* élevé) peut faire en sorte que l'erreur soit sous-estimée par la métrique de variation relative maximale (*VarRelMax*) et que les itérations soient terminées prématurément. À l'inverse, lorsqu'une grande précision est requise, la solution obtenue lorsque les itérations NR sont terminées est plus proche de la solution exacte. Par conséquent, la proportion de cas de convergence lente causant des résultats erronés est réduite au coût d'un nombre plus élevé d'itérations.

Les résultats expérimentaux montrent que la méthode proposée diminue le nombre d'itérations requises de 20 à 40 % en moyenne, selon les cas étudiés. La proportion de résultats erronés n'est que de 0.2 à 0.3 % pour des circuits de référence légèrement modifiés, alors qu'elle augmente à 2.5 à 5.7 % pour des circuits sévèrement altérés. D'ailleurs, cinq des six circuits de référence étudiés présentaient moins de 1 % de résultats erronés, peu importe les modifications effectuées et la précision requise.

La méthode d'augmentation de la tolérance relative a été présentée dans [42]. Elle peut être utilisée par n'importe laquelle des applications ciblées par ce mémoire : simulation de pannes, dimensionnement de circuits et analyse Monte Carlo. Le seul paramètre nécessaire est la précision requise par l'application. Un compromis entre le temps de calcul et la précision des résultats est réalisé. De plus, la métrique de variation relative *VarRelMax* sera réutilisée dans la méthode appelée *Threshold-Based Simulation Accuracy* (*TBSA*) qui est proposée au chapitre 4 afin d'approximer la précision atteinte à chaque itération.

CHAPITRE 3

Mise en contexte des articles proposés

Le chapitre précédent proposait une méthode permettant de terminer les itérations NR avant la convergence lorsque la précision requise est atteinte. Les deux chapitres qui suivent proposent d'autres techniques permettant de réduire le temps de calcul nécessité par les simulations multiples. Ils sont constitués d'articles soumis à des revues spécialisées.

Le premier article, présenté au chapitre 4, propose une nouvelle méthode d'arrêt des itérations NR avant la convergence qui s'adresse à la simulation de pannes analogiques. Cet article a été soumis en septembre 2005 à la revue *Journal of Electronic Testing : Theory and Applications* et accepté pour publication en juin 2006. Certaines notions de base en simulation analogique ainsi que les méthodes existantes permettant d'accélérer la simulation DC, qui ont été abordées au chapitre 1, y sont reprises. L'analyse de la convergence DC présentée au chapitre 2 y est également résumée. La métrique de variation relative maximale, qui permet d'approximer la précision à une itération donnée, est d'ailleurs réutilisée. Par la suite, une nouvelle méthode permettant de terminer les itérations avant la convergence dans une application de simulation de pannes, appelée *TBSA*, est proposée. Tout comme la méthode d'augmentation de la tolérance relative présentée au chapitre 2, cette nouvelle méthode nécessite une AIPS provenant d'un circuit similaire. Cette fois, les itérations sont terminées dès que la tension de sortie obtenue à une itération donnée est suffisamment précise pour permettre la classification de la panne comme détectée ou non. À cette fin, la métrique *VarRelMax* (nommée *WorstRelVar* dans l'article) permet d'approximer la précision de la solution à chaque itération et ce, dans le but de prévoir si la panne peut être classifiée immédiatement avec un faible risque d'erreur.

Pour sa part, le second article, présenté au chapitre 5, propose un simulateur de type SPICE optimisé pour effectuer rapidement les simulations multiples. En effet, les simulateurs existants ne sont pas conçus pour être appelés des milliers de fois pour un circuit légèrement modifié à chaque fois, puisque plusieurs étapes sont répétées. De plus, les deux méthodes d'arrêt des itérations avant la convergence proposées dans ce mémoire ne peuvent être utilisées avec des simulateurs conventionnels. C'est pourquoi un simulateur rapide pour les applications visées par ce mémoire a été développé. L'article décrivant ce simulateur a été soumis à la revue « *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* » en novembre 2006. On y présente tout d'abord les caractéristiques communes aux applications nécessitant des simulations multiples, telles que vues au chapitre 1 de ce mémoire. Les améliorations suivantes sont par la suite proposées : 1) une connexion *socket* avec l'application appelante, 2) l'injection efficace des modifications dans le circuit nominal, 3) l'utilisation d'approximations initiales précises des solutions et 4) l'utilisation de méthodes d'arrêt de la simulation DC avant la convergence. Il est possible d'obtenir plus de détails sur le fonctionnement des *sockets* dans [50]. La méthode d'augmentation de la tolérance relative du chapitre 2 est reprise dans l'article proposé, alors que la méthode *TBSA* du chapitre 4 est généralisée à toutes les applications qui nécessitent la comparaison entre deux circuits, par exemple le dimensionnement de circuits. Toutes les fonctionnalités proposées ont été incorporées dans un simulateur analogique nommé *MultiSPICE* et basé sur le logiciel *SPICE3f5* [49]. Afin de valider l'environnement de simulation offert par *MultiSPICE*, les trois applications mentionnées dans ce mémoire ont été développées : simulation de pannes, dimensionnement automatique de circuits et analyse Monte Carlo. Celles-ci offrent une interface graphique utilisateur et tirent partie des fonctionnalités offertes par le simulateur *MultiSPICE*.

CHAPITRE 4

Article - TBSA: Threshold-Based Simulation Accuracy Method for Fast Analog DC Fault Simulation

Michel Morneau, Abdelhakim Khouas
Electrical engineering department, École Polytechnique de Montréal

Abstract: Starting from a good solution approximation has proved to be very efficient to reduce CPU time required by DC simulation of analog circuits. In order to obtain an additional speedup in DC fault simulation, this paper proposes a new criterion to end the Newton-Raphson (NR) iterative algorithm before convergence. In the case where an initial solution approximation is used, the analysis of the NR algorithm behavior until convergence is presented and a Threshold-Based Simulation Accuracy (*TBSA*) method is then proposed. *TBSA* stops the iterations when the solution at current NR iteration is sufficiently accurate to immediately classify the fault. According to the detection thresholds, a CPU time/accuracy tradeoff is achieved without altering the fault classification results. The proposed method has been validated on 12 MOS and BJT benchmark circuits considering DC fault simulation under process parameter variations. *TBSA* is compared to two existing methods which are: standard simulation until convergence method which is accurate but requires a large CPU time, and single NR iteration method which is very fast but without any control over the accuracy. All the compared methods reuse the fault-free circuit results as initial solution for each faulty circuit simulation. It is shown that *TBSA* requires an intermediate number of NR

iterations while achieving correct fault classification, especially for parametric faults which take advantage of using a more accurate initial solution.

Keywords: analog testing, DC fault simulation, analog fault detection, Newton-Raphson algorithm

4.1 Introduction

During the last decade, the development of systems on chip allowed the integration of analog and digital circuits on the same die to reduce manufacturing costs. The analog parts are used to communicate with the environment which is analog in many applications, while the processing is done by the digital parts. Since the early 1990s, the average growth of such mixed-signal integrated circuits has been between 15 and 20 % per year [1]. To support this growth, the development of efficient testing methodologies and tools is of utmost importance.

Even though efficient digital testing tools exist today, fault simulation in the analog domain has not reached this level of maturity [2]. Modeling defects using fault models is much more complicated; both catastrophic (hard) and parametric (soft) faults must be considered and are usually related with the circuit layout [3]. Since each faulty circuit must be simulated, analog fault simulation is often the bottleneck in analog and mixed-signal testing [4].

Faults are generally simulated sequentially, using an inner-loop commercial simulator like HSPICE [5] or SPECTRE [6], without taking advantage of the similarities between fault-free and faulty circuits. For medium and large analog circuits, the CPU time may be intractable if a comprehensive test set is used. DC simulation [7] is the least expensive method and can only detect the faults affecting the operating point, while other faults

require AC [8] or transient simulation [4]. This paper focuses on DC simulation for fast detection of easy-to-detect parametric and catastrophic faults.

4.1.1 Previous Works

Linear analog circuits have first been addressed. Sensitivity computation using the adjoint network technique has been incorporated in a test tool in [9]. The construction of a state description to represent the fault-free circuit and its reutilization by faulty circuits has been studied in [10]. In [11], admittance stamps are used to inject faults directly into the matrix representation of fault-free circuit.

However, for nonlinear circuits, efficient simulation methods are more difficult to develop. Simpler models can be constructed from piecewise linear approximations (PWL) [12], SPICE macromodels [13], and hardware description languages (HDL) [14]. Nevertheless, the generation of these models is a hard task and, since they neglect the second-order phenomena, they cannot guarantee valid results.

The Newton-Raphson (NR) iterative algorithm implemented in SPICE simulators is highly sensitive to the initial solution. Thus, DC fault simulation is considerably accelerated if a judicious approximation of the final solution (e.g. fault-free circuit solution) is used as initial solution of each faulty circuit. The faults can also be ordered in such a way that a previous faulty circuit solution provides initial solution to the current faulty circuit. The authors in [7] perform one-step NR iteration using Householder's formula to approximate the faulty responses and use a greedy algorithm to order the faulty circuits. A first order interpolation from consecutive previous solutions to set the initial solution of parametric faults is also explored in [4]. To approximate the fault coverage, a single NR iteration using Householder's formula with initial solution set to fault-free solution was proposed in [15]. A similar method was also studied in DC test

generation [16]. However, in [15] and [16], no accuracy control is performed, while in [4] and [7], full accuracy simulations require more CPU time.

4.1.2 Contributions of this Work

The aim of this paper is to propose some tradeoff between full accuracy and single-iteration DC simulation when initial guess solutions are used. The NR algorithm is studied in order to analyze its convergence behavior in DC simulation. A new heuristic for DC fault simulation, called *Threshold-Based Simulation Accuracy* (*TBSA*), is therefore proposed. *TBSA* allows ending the NR algorithm before convergence once the accuracy is sufficient to correctly classify the simulated fault.

This paper is organized as follows: Section 4.2 studies the NR algorithm in DC simulation and proposes a metric to approximate the intermediate solution accuracy at each iteration. The considered fault detection algorithm which takes into account process variations is presented in Section 4.3. Section 4.4 describes the proposed *TBSA* method. Section 4.5 shows and discusses the experimental results for parametric and catastrophic faults on 12 benchmark circuits. In Section 4.6, we present our conclusions.

4.2 Analysis of Newton-Raphson Algorithm

SPICE simulators use NR iterative algorithm to solve the systems of nonlinear equations. Neglecting the time spent to parse the netlist file and setup the circuit representation, the CPU time involved in DC simulation is proportional to the number of NR iterations. Reducing the number of iterations decreases in the same way the CPU time. This section analyses the iterations in order to approximate the exact solution before final convergence of the NR algorithm.

4.2.1 Basis of DC Simulation

NR iterations start from a solution approximation, the circuit components are linearized around this solution using the Modified Nodal Analysis (MNA) representation [17], and a system of linear equations is then obtained. The solution of this system is considered as new approximation, hopefully closer to the exact solution, and is used for the next iteration. Equation (4.1) represents the k^{th} NR iteration, where J is the Jacobian matrix (also called linearized conductances), x^{k-1} is the previous solution approximation, x^k is the new solution approximation, and F is the function to minimize. Both J and F are built during the component linearization phase from the equations included into the encapsulated component models.

$$J(x^{k-1})(x^k - x^{k-1}) = -F(x^{k-1}) \qquad (4.1)$$

The linear components always have constant values in the Jacobian and thus, a single iteration is required for linear circuits. For nonlinear circuits, the convergence is reached when the solutions obtained during two consecutive NR iterations differ by less than a given tolerance. As shown in equation (4.2), the tolerance tol_i^k for the node i at iteration k depends on the absolute (*abstol*) and relative (*reltol*) tolerances and on the solution approximation of this node at current (x_i^k) and previous (x_i^{k-1}) iterations. The convergence criterion used in *SPICE3f5* [18] is shown in equation (4.3).

$$tol_i^k = abstol + reltol \times \max\left(|x_i^k|, |x_i^{k-1}|\right) \qquad (4.2)$$

$$if \quad |x_i^k - x_i^{k-1}| \le tol_i^k \ \forall i \quad \rightarrow \quad convergence \qquad (4.3)$$

Default values used in SPICE simulators [18] and [6] are *reltol* = 10^{-3}, *abstol* = 10^{-6} V for voltages and *abstol* = 10^{-12} A for currents. *abstol* can be neglected except for values x_i^k close to 0. If a voltage or current x_i^k is modified by more than its tolerance tol_i^k between two consecutive iterations, a new iteration is performed. Let us consider ε_i^k the

approximation error for the node i at k^{th} iteration, if the approximated solution is close to the exact solution, we have:

$$\lim_{k\to\infty} \frac{\left|\varepsilon_i^{k+1}\right|}{\left|\varepsilon_i^{k}\right|^2} = constant \qquad (4.4)$$

Equation (4.4) shows that NR algorithm exhibits quadratic convergence when the approximated solution is close to the exact solution [17]. For example, if *constant* = 1 and $\varepsilon_i^k = 0.1$, one can expect $\varepsilon_i^{k+1} = 0.01$ and $\varepsilon_i^{k+2} = 0.0001$ for $(k+1)^{th}$ and $(k+2)^{th}$ iteration when the approximated solution is close to the exact solution. However, if the approximation is far from the exact solution, no convergence assertion can be made and additional iterations are needed before reaching convergence.

4.2.2 DC Convergence Analysis

Using *SPICE3*, a typical convergence case with no initial solution is illustrated in Figure 4.1 (a). The DC simulation starts with a solution where all voltages are set to 0 V. The output voltage oscillates during a certain number of iterations before converging to its final value. The first 7 iterations can be considered as "lost CPU time" since the algorithm is not converging. However, at iteration 8, an approximated value which is close to the exact value is obtained. The convergence is then reached with a few additional iterations.

Figure 4.1 (b) shows the convergence behavior for the same circuit but taking advantage of an initial solution provided by a similar circuit with some altered parameters. The convergence is reached in 4 iterations, without significant voltage oscillations. The system of nonlinear equations describing the circuit remains stable and converges rapidly. Reusing the solution of a similar circuit reduces the number of iterations and accelerates the convergence by immediately providing an approximated solution which is close to the

exact solution where quadratic convergence is expected. In analog fault simulation, the solution of the fault-free circuit or a previous faulty circuit can be reused as an initial solution, which allows reducing the number of NR iterations without altering the accuracy [7].

In order to achieve an additional reduction of the number of NR iterations, the simulation could be ended once a sufficient accuracy is obtained. Figure 4.2 (a) shows the output approximation error which decreases very quickly and exhibits near to quadratic convergence. Since the output approximation error is unknown during simulation, another indicator must be used to detect if the system is converging. This new metric, called "Worst Relative Variation" (*WorstRelVar*), is the largest relative variation affecting the circuit nodes between two consecutive iterations k-1 and k, as shown in equation (4.5).

$$WorstRelVar^k = \max_{\forall i}\left(\frac{|x_i^k - x_i^{k-1}|}{\max(|x_i^k|, |x_i^{k-1}|, abstol)}\right) \qquad (4.5)$$

Figure 4.2 (b) illustrates this metric for the previous example. Like the output error, $WorstRelVar^k$ decreases rapidly, but remains larger than the error at each iteration, as can be seen in Figure 4.2. The idea is to interpret $WorstRelVar^k$ as the accuracy achieved by the intermediate solution x_i^k, so the error is assumed never to be larger than this metric. This is the case during quadratic convergence since the error decreases by more than one order between consecutive iterations. There is a strong relation between the convergence criterion in equations (4.2)-(4.3) and *WorstRelVar*: if at any iteration k, $WorstRelVar^k$ is smaller than *reltol*, convergence is assumed and the iterations are ended.

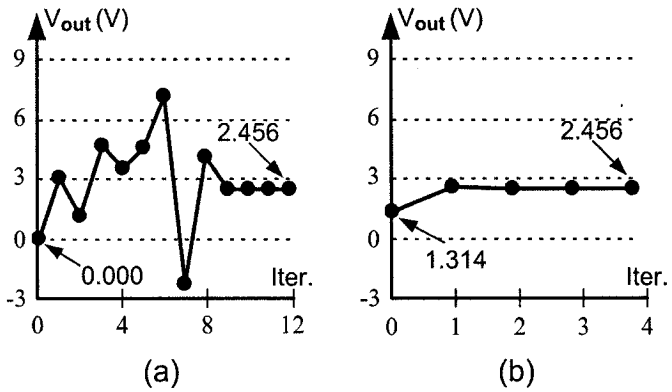


Figure 4.1 : Output voltage values during NR iterations: (a) with and (b) without initial solution.

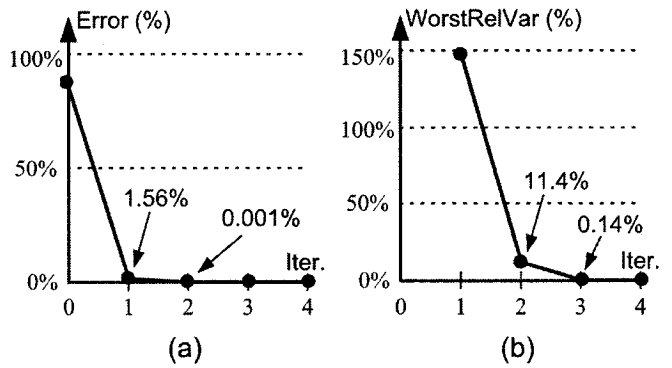


Figure 4.2 : (a) Output approximation error and (b) WorstRelVar metric during NR iterations with initial solution

However, if the initial solution is inaccurate, the convergence is not straightforward. Large oscillations may occur like in Figure 4.1 (a) and cause a hard convergence case. In such a case, $WorstRelVar^k$ is not guaranteed to be a valid accuracy approximation since the behavior of the nonlinear system may become unpredictable. Although the correctness of $WorstRelVar^k$ is not guaranteed, in most cases this metric is a valid approximation of the current accuracy [19].

The *WorstRelVar* metric suggests relaxing *reltol* SPICE parameter to the needed accuracy in order to reduce the number of NR iterations. In our previous work [19],

analog DC simulations taking advantage of initial solutions and relaxed values of *reltol* have shown satisfying results for *reltol* increased up to 20 %. Moreover, two important factors affect the reliability of the *reltol* relaxation method [19]:

1) *Circuit behavior alteration*: Modifications causing small effects on the circuit behavior favor a more accurate initial solution, since the exact solution is more easily predictable. The occurrence of hard convergence cases and consequently the risk of obtaining erroneous results caused by these cases are thus reduced.

2) *Needed accuracy*: With high accuracy (small *reltol*), NR iterations are stopped when the algorithm is close to the convergence. Therefore, the risk of hard convergence cases causing erroneous results is reduced at the cost of a larger number of iterations. On the other hand, low accuracy may cause the error to be underestimated by $WorstRelVar^k$ metric and the iterations may be stopped prematurely.

In DC fault simulation, the *WorstRelVar* metric will be reused to predict the minimal and maximal expected values for the faulty circuit solution, which are required by the *TBSA* method proposed in Section 4.4.

4.3 Fault Detection under Process Parameter Deviations

During DC fault simulation, a fault is classified as detectable or not detectable, depending on how close the faulty circuit response is from the fault-free circuit response. Three methods are commonly used to set the output detection thresholds:

1) Absolute deviation value (e.g. ±50 mV).

2) Percentage of the fault-free output value (e.g. ±5 %).

3) Factor of the fault-free distribution of the output value (e.g. ±2*standard deviation of the output).

The two first methods are straightforward and the thresholds can be defined immediately after a single simulation of the fault-free circuit. If a faulty circuit response is not within

the thresholds, the fault is classified as detectable. However, fault masking due to process variations cannot be handled by those methods [8].

In order to consider the process random parameter deviations, the third method is employed. Monte-Carlo simulations are required to compute the distributions of the good (fault-free) circuit (D_g) and the faulty circuit (D_f) at the considered output. For a normal (Gaussian) distribution, the probability that a continuous random variable V has a value within a certain interval is given by the area under the distribution curve in this interval. If μ is the mean and σ the standard deviation of a given output distribution, this probability is 95 % for the interval $[\mu\text{-}2\sigma, \mu+2\sigma]$. If there is no overlap between the distributions D_g and D_f, the fault f is considered detectable; if D_f is included in D_g the fault is not detectable; otherwise the fault is partially detectable. Equation (4.6) presents the detectability criterion.

$$|\mu_g - \mu_f| > 2\sigma_g + 2\sigma_f \quad \rightarrow \quad \textit{Fault is detectable} \tag{4.6}$$

If one avoids performing costly Monte-Carlo simulations on faulty circuits, the mean and standard deviation values become unknowns for the faulty circuits. However, a single simulation generates an output value V_f within the interval $[\mu_f\text{-}2\sigma_f, \mu_f+2\sigma_f]$ of the corresponding distribution with 95 % confidence. Therefore, assuming $\sigma_f = \sigma_g$, criterion (4.6) becomes (4.7) or equivalently, (4.8) [20]:

$$|\mu_g - V_f| > 6\sigma_g \quad \rightarrow \quad \textit{Fault is detectable} \tag{4.7}$$

$$\begin{cases} V_f < \mu_g - 6\sigma_g \\ V_f > \mu_g + 6\sigma_g \end{cases} \rightarrow \quad \textit{Fault is detectable} \tag{4.8}$$

The inequalities in criterion (4.8) remain valid if $\sigma_f < \sigma_g$. A graphical interpretation of criterion (4.8) is shown in Figure 4.3. For an arbitrary faulty output response V_f, the minimal ($D_{f,min}$) and maximal ($D_{f,max}$) expected faulty distributions, respectively centered

at $\mu_{f,min} = V_f - 2\sigma_g$ and $\mu_{f,max} = V_f + 2\sigma_g$, are presented and compared to the fault-free distribution D_g. In Figure 4.3 (a), the faulty circuit response V_f is far from the mean value μ_g of the fault-free circuit. Since the distributions $D_{f,min}$ and $D_{f,max}$ cannot overlap D_g, the fault is classified as detectable. However, in Figure 4.3 (b), $D_{f,min}$ overlaps D_g, so such fault is considered undetected and more simulations are needed to determine if the fault is detectable or partially detectable [20].

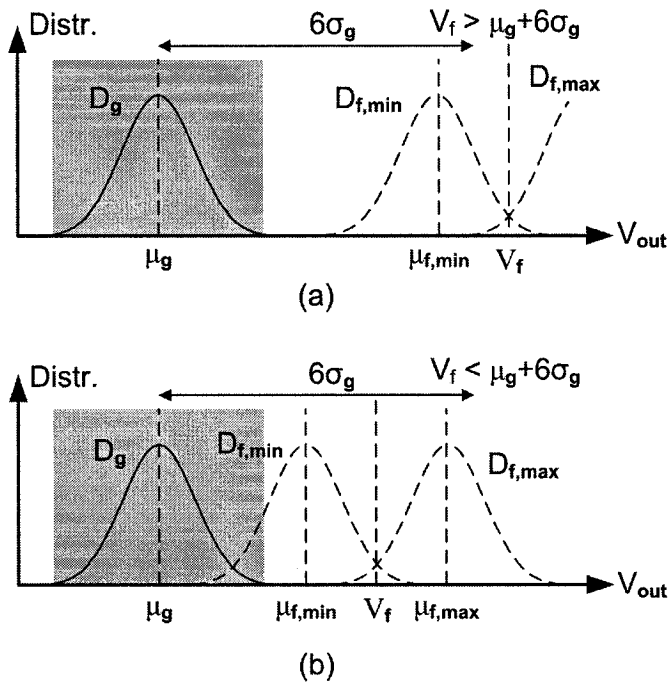


Figure 4.3 : Comparing distributions of fault-free and faulty circuit output response: (a) detected fault, and (b) undetected fault.

This detection criterion has been experimented in [20], with using the detection probabilities for the faults that are classified partially detectable. Nevertheless, since the fault simulations only serve to evaluate criterion (4.8), accurate results are not required if the use of approximated faulty solutions is sufficient to correctly classify the faults.

4.4 Threshold-Based Simulation Accuracy (*TBSA*) Method

In this section, a new criterion to stop NR iterations before convergence while achieving correct fault classification is presented. The *WorstRelVar* metric introduced in Section 4.2 will be exploited to stop the iterations once the solution at current iteration is enough accurate to correctly classify the fault. The use of a linear interpolation to generate an initial solution for the simulation of the faulty circuits instead of simply reusing the fault-free circuit solution is also explored.

4.4.1 TBSA Algorithm

Injecting a fault into a nominal circuit causes a deviation on its DC operating point. If the fault-free circuit solution is reused as an initial solution for the DC simulation of the faulty circuit, three situations may occur:

1) If the fault does not cause significant deviations on the circuit behavior, the convergence is fast and the fault can be rapidly classified as not detectable.

2) If an easily detectable fault is simulated, hard convergence often occurs, but the deviation is so large that accurate simulation results are not required to classify the fault as detectable.

3) If the fault causes a response deviation close to a detection threshold, higher accuracy is needed to correctly classify the fault according to the detection criterion (4.8).

The preceding remarks suggest tolerating higher error on the simulated solution of the faulty circuit when the exact output value is expected to be far from the detection thresholds. The simulation is stopped once the accuracy at current iteration allows classifying the fault. If the approximated solution is close to the detection thresholds, the simulation is continued to improve the accuracy and hence correctly classify the fault.

As stated in Section 4.2, the accuracy at current iteration can be reasonably approximated by the $WorstRelVar^k$ value computed according to equation (4.5). Hence, $WorstRelVar^k$ is considered as the maximal relative variation expected between the approximated solution at current iteration and the final solution at convergence. Consequently, the minimal ($V_{f,min}$) and maximal ($V_{f,max}$) expected values of the exact response V_f can be computed from the approximated response at current iteration V_f^k according to equation (4.9):

$$\begin{aligned} V_{f,\min}{}^k &= \left(1 - WorstRelVar^k\right) V_f{}^k \\ V_{f,\max}{}^k &= \left(1 + WorstRelVar^k\right) V_f{}^k \end{aligned} \qquad (4.9)$$

The exact faulty output response is therefore assumed to be within the range [$V_{f,min}$, $V_{f,max}$]. Knowing the minimal and maximal values of the exact response, it is possible to compare their position to the fault detection thresholds defined by equation (4.8) in previous section. At each NR iteration, one of the following cases occurs:

1) Both $V_{f,min}$ and $V_{f,max}$ are below ($V_{f,max} < \mu_g$-$6\sigma_g$) or above ($V_{f,min} > \mu_g$+$6\sigma_g$) the detection range [μ_g-$6\sigma_g$, μ_g+$6\sigma_g$] defined by criterion (4.8). In this case, the exact solution V_f is also expected to be outside the range. The fault is considered detectable and the NR iterations are stopped.

2) Both $V_{f,min}$ and $V_{f,max}$ are within the detection range [μ_g-$6\sigma_g$, μ_g+$6\sigma_g$]. The exact solution V_f will remain inside this range, thus the fault can be immediately classified as not detectable and the NR iterations are stopped.

3) $V_{f,min}$ and $V_{f,max}$ are on each side of the detection range (one is within the range [μ_g-$6\sigma_g$, μ_g+$6\sigma_g$] while the other is outside the range). In this case, the current accuracy is not sufficient to classify the fault and the NR iterations are continued.

Figure 4.4 expresses the complete *Threshold-Based Simulation Accuracy* (*TBSA*) algorithm considering a circuit with single output.

1 Define the DC input stimulus to apply
2 Perform Monte-Carlo simulations on fault-free circuit
3 Compute μ_g and σ_g for the output node
4 **for** Each fault f_j in the fault list (j = 1...m) **do**
5 Set initial solution x_j^0
6 k = 0
7 stop ← false
8 **repeat**
9 k = k+1
10 Perform the k^{th} NR iteration to obtain x_j^k
11 V_{fj}^k ← Output response from x_j^k
12 $$\textit{WorstRelVar}^k = \max_{\forall i}\left(\frac{|x_i^k - x_i^{k-1}|}{\max(|x_i^k|, |x_i^{k-1}|, \textit{abstol})}\right)$$
13 $V_{fj,min}^k = (1 - \textit{WorstRelVar}_j^k) * V_{fj}^k$
14 $V_{fj,max}^k = (1 + \textit{WorstRelVar}_j^k) * V_{fj}^k$
15 **if** ($V_{fj,min}^k > V_{fj,max}^k$) **then** *// when $V_{fj}^k < 0$*
16 Swap ($V_{fj,min}^k$, $V_{fj,max}^k$)
17 **end if**
18 **if** ($V_{fj,max}^k < \mu_g - 6\sigma_g$) **then** *// below range*
19 Fault f_j is classified as detectable
20 stop ← true
21 **else if** ($V_{fj,min}^k > \mu_g + 6\sigma_g$) **then** *// above range*
22 Fault f_j is classified as detectable
23 stop ← true
24 **else if** (($V_{fj,min}^k \geq \mu_g - 6\sigma_g$) **and** ($V_{fj,max}^k \leq \mu_g + 6\sigma_g$)) **then** *// inside range*
25 Fault f_j is classified as not detectable
26 stop ← true
27 **end if**
28 **if** (convergence reached) **then**
29 Classify fault f_j
30 stop ← true
31 **end if**
32 **while** (stop = false)
33 **end for**

Figure 4.4 : *TBSA* algorithm

Figure 4.5 illustrates some fault simulation examples using *TBSA* method. For each example, V_f is the exact faulty response, V_f^k is the approximated response computed at k^{th} iteration and $V_{f,min}^k$ and $V_{f,max}^k$ are the minimal and maximal bounds of V_f computed at k^{th} iteration according to equation (4.9). In Figure 4.5 (a), at iterations 1 and 2, $V_{f,max}$ is inside the detection range $\mu_g \pm 6\sigma_g$ while $V_{f,min}$ is outside the range, so the iterations are carried on. However, at iteration 3, both $V_{f,min}$ and $V_{f,max}$ are below the thresholds, so the fault is classified as detectable and the simulation is ended. In Figure 4.5 (b), since both $V_{f,min}$ and $V_{f,max}$ are inside the range at iteration 3, the simulation is ended and the fault is classified as not detectable. Figure 4.5 (c) presents a situation where the faulty output response is so far from the detection range that a single iteration is sufficient to classify the fault as detectable. On the other hand, in Figure 4.5 (d), the faulty output response is very close to the detection threshold and more iterations are performed to increase the accuracy and hence correctly classify the fault.

The faults causing large deviations are usually those that require a large number of iterations when starting from the fault-free solution. For such faults, *TBSA* ends the simulation even if much more iterations are required to obtain a more accurate solution, saving substantial CPU time.

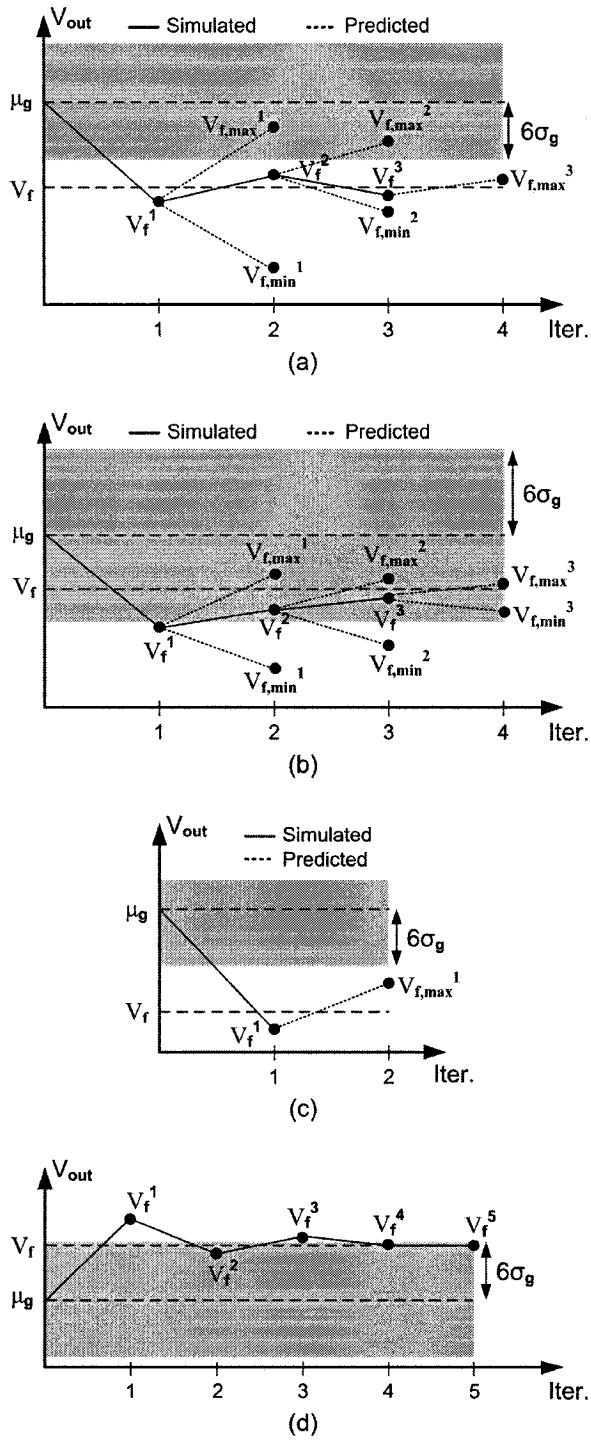


Figure 4.5 : Fault simulation examples using *TBSA* : (a) the fault is classified as detectable at iteration 3; (b) the fault is classified as not detectable at iteration 3; (c) single iteration is sufficient to classify the fault as detectable; (d) more iterations are required before fault classification

4.4.2 Initial Solution Interpolation

In the previous subsection, the fault-free circuit solution is directly reused as initial solution of each faulty circuit simulation. However, for parametric faults associated to a variation of single device parameter, the generated faulty circuits can be easily ordered in two lists starting from the fault-free circuit. The faulty circuits with faulty parameter value larger than the fault-free value are ordered in an increasing way according to the faulty parameter value, while the remaining faulty circuits are ordered in a decreasing way, as suggested in [4]. The initial solution of each faulty circuit is then generated by performing a linear interpolation of the two previous computed solutions. Equation (4.10) presents the initial solution interpolation x_j^0 for the fault j in a given fault list in which x_0 is the fault-free solution.

$$\begin{aligned} x_1^{\,0} &= x_0 \\ x_j^{\,0} &= x_{j-1} + (x_{j-1} - x_{j-2}) \quad (j \geq 2) \end{aligned} \tag{4.10}$$

Figure 4.6 shows an example of initial solution interpolation for a resistor with fault-free value R and faulty values 1.1*R, 1.2*R and 1.4*R.

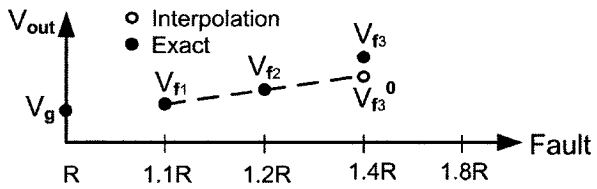


Figure 4.6 : Initial solution generated by linear interpolation

During full accuracy simulation, the solution interpolation reduces the number of NR iterations required since the faults in a list lie on the homotopy curve of the corresponding parameter [21]. However, *TBSA* ends the iterations before convergence, thus the previous faulty solutions may be not enough accurate and can affect the correctness of the interpolated solution. Therefore, experimental results are required to

evaluate *TBSA* performances when the initial faulty circuit solutions are interpolated from previous *TBSA* results.

4.5 Experimental Results

The *SPICE3f5* [18] simulator has been adapted to validate the proposed method. Transistor models and computations that perform NR iterations remain the same as the original software. A new data structure has been added to store and reuse the solutions of fault-free and faulty circuits, allowing linear solution interpolation. *TBSA* has been included as another criterion to end the NR iterations if convergence is not reached after current iteration. The faults are directly injected in the fault-free circuit MNA system from custom commands in the netlist file. Default *SPICE3f5* continuation techniques G_{min} stepping and source stepping [21] are exploited if convergence is not reached after 100 iterations (*itl1* = 100).

4.5.1 Fault Modeling

Inductive fault analysis (IFA) is recommended to generate a realistic set of faults [22], especially for catastrophic faults. In the absence of circuit layout and process information, the fault dictionary is built from the netlist components. Parametric faults are injected in the fault-free circuit MNA system by altering the nominal value of the parameters to their faulty values. The following parameters are considered for parametric faults: resistance, MOS transistors width and length and BJT area factor. For each of these parameters, 9 different faulty values are simulated: 25 %, 40 %, 60 %, 80 %, 125 %, 150 %, 200 %, 300 % and 400 % of the nominal value.

Short faults in resistors and capacitors are modeled as small short resistance between their terminals. In MOS transistors, drain-gate, gate-source and drain-source shorts are considered while in BJT transistors, collector-base, base-emitter and collector-emitter

shorts are injected. Short resistances typically vary from 1 to 500 Ω [8]. Both values are simulated as distinct faults. Open faults are injected in resistors by setting their resistance to 10 MΩ. Since capacitors do not conduct in DC simulation, open faults are not considered in capacitors. Opens in MOS and BJT transistors are modeled by forcing the transistor to remain in the cut-off region as suggested in [23]. To avoid floating gates, 10 MΩ resistors are added between their terminals. Figure 4.7 shows the faults injected in a MOS transistor and Table 4.1 summarizes the injected faults.

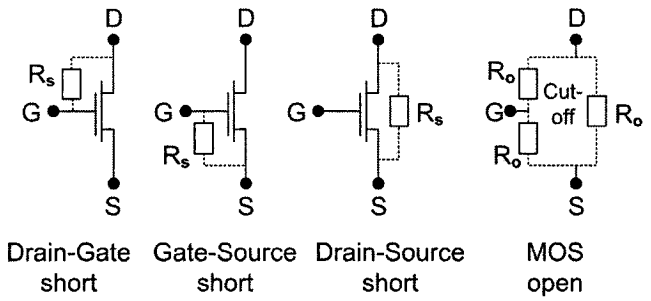


Figure 4.7 : Catastrophic fault models for MOS transistors (Rs=1Ω, 500Ω; Ro=10MΩ)

Table 4.1 : Faults injected in the benchmark circuits

Dev.	Type	Value
Res.	Short	R_S=1Ω, 500Ω
	Open	R_O=10MΩ
	Parametric (Resist.)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%
Cap.	Short	R_S=1Ω, 500Ω
MOS	Short G-S, D-S, D-G	R_S=1Ω, 500Ω
	Open	Cut-off, R_O=10MΩ
	Parametric (W and L)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%
BJT	Short B-E, C-E, C-B	R_S=1Ω, 500Ω
	Open	Cut-off, R_O=10MΩ
	Parametric (Area)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%

4.5.2 Experimental Circuits

The two first experimental circuits come from ITC'97 [24] while the ten others are from Circuitsim90 [25] sets of benchmark circuits. Characteristics and number of injected faults in each circuit are summarized in Table 4.2. Circuit names are in column 1, while the transistor models in the circuits are presented in column 2. The number of injected parametric and catastrophic faults appears in columns 3 and 4 respectively. Monte-Carlo simulations are performed on the fault-free circuits by randomly modifying the parameters introduced in Table 4.1 according to a uniform distribution. Monte-Carlo simulation results revealed that some benchmark circuits are more sensitive to the process parameter variations than others. Since no process information is available and the main objective is to obtain reasonable detection thresholds to validate *TBSA* method, random variations are defined in such a way to observe a standard deviation σ_g between 1.5 and 4 % of the mean value. For 1000 Monte-Carlo simulations on the fault-free circuit, columns 5 and 6 in Table 4.2 show the corresponding output voltage mean value μ_g and standard deviation σ_g in percentage of μ_g respectively. The DC input stimulus of each benchmark circuit is set to a value that favors a good DC fault coverage.

Table 4.2 : Characteristics of the experimental circuits

Circuit	Transistor model	# faults		Mean, standard deviation	
		Parametric	Catastrophic	μ_g (V)	σ_g (%)
leapfrog	mos1	1089	437	1.32	3.87
continous	mos3	574	222	-2.00	1.89
astabl	bjt	54	30	1.03	2.88
bias	bjt	162	106	-0.04	1.55
latch	bjt	216	128	2.07	2.72
vreg	bjt	269	169	2.30	1.72
ab_integ	mos2	585	274	0.08	1.73
ab_opamp	mos2	594	240	2.50	1.56
gm6	mos2	90	35	1.15	2.38
schmitslow	mos2	144	56	0.52	3.46
gm2	mos3	109	59	3.23	2.19
gm3	mos3	539	211	0.27	3.62

4.5.3 Results and Discussions

The benchmark circuits allow comparing the performances of the four different DC fault simulation methods addressed in this paper. In the following tables, they are identified as:
1) *Spice3*: The standard *SPICE3f5* algorithms with no initial solution ($x^0 = 0$ for each node). This is similar to using commercial simulators [5] and [6].
2) *Init.-sol.*: An initial solution is available for each faulty circuit and the simulation is conducted until NR algorithm convergence.
3) *TBSA*: The proposed *Threshold-Based Simulation Accuracy* method described in Section 4.4.
4) *1-iter.*: A single iteration is performed using an initial solution, similar to [15] and [16].

It must be mentioned that the faulty circuits that cannot converge under *Init.-sol.* method and *SPICE3f5* continuation techniques have been dropped since the exact fault classification remains unknown.

Table 4.3 and Table 4.4 present the DC simulation results for parametric faults. For each method except *Spice3*, the fault-free circuit solution is reused as initial solution for each faulty circuit. Table 4.3 details the number of NR iterations and iteration gain for each of the studied methods. Columns 2 to 5 show the average number of NR iterations performed per faulty circuit. The large number of iterations for some circuits is due to the frequent requirement to *SPICE3f5* continuation techniques to obtain a solution. The iteration gain (columns 6-8) of each method is defined as the number of iterations required by *Spice3* divided by the number of iterations required by the considered method. The last line presents the average of each performance value for the 12 studied circuits.

Table 4.3 : Iteration gain for parametric faults reusing fault-free solution as initial solution

Circuit	Average number of iterations				Iteration gain		
	Spice3	Init.-sol.	TBSA	1-iter.	Init.-sol.	TBSA	1-iter.
leapfrog	42.4	3.97	2.41	1	10.7	17.6	42.4
continous	123	5.86	2.91	1	21.0	42.3	123
astabl	9.94	4.83	1.57	1	2.06	6.33	9.94
bias	54.3	3.72	1.74	1	14.6	31.2	54.3
latch	25.3	3.65	1.94	1	6.93	13.0	25.3
vreg	383	9.36	1.32	1	40.9	290	383
ab_integ	18.9	5.01	1.81	1	3.77	10.4	18.9
ab_opamp	394	4.41	2.17	1	89.3	182	394
gm6	12.2	5.73	1.97	1	2.13	6.19	12.2
schmitslow	22.5	14.3	3.92	1	1.57	5.74	22.5
gm2	8.56	4.65	2.41	1	1.84	3.55	8.56
gm3	10.8	2.76	1.55	1	3.91	6.97	10.8
Average	92.1	5.69	2.14	1.00	16.2	43.0	92.1

It can be seen from Table 4.3 that the use of an initial solution reduces substantially the number of iterations over the usual SPICE simulation. The average number of iterations has dropped by a factor of 16.2 from 92.1 for *Spice3* to 5.69 for *Init.-sol. TBSA* method requires an average of 2.14 iterations per faulty circuit. Hence, *TBSA* reduces the number of iterations by a factor of 43.0 (92.1/2.14) over usual SPICE simulation and by a factor of 2.66 (5.69/2.14) over *Init.-sol.* method. The fastest method is *1-iter.*, which has an average number of iterations 50 % less than *TBSA*, but as it is shown in Table 4.4, this method is less efficient in terms of fault detection and classification.

The parametric fault detection results for each of the studied methods are detailed in Table 4.4. Columns 2-4 show the number of detected faults by each method according to the detection criterion (4.8). Nevertheless, the number of detected faults is not the best measure for the efficiency of each method. Indeed, if a detectable fault f_i is falsely classified as detected while another undetectable fault f_j is classified as detected; those errors cancel each other and show a correct number of detected faults while two faults are misclassified. For this reason, the number of fault misclassifications for each method is

reported in columns 5 and 6. The percentage of misclassification is defined as the number of fault misclassifications divided by the exact number of detected faults and is presented in the two last columns.

Table 4.4 : Fault detection for parametric faults reusing fault-free solution as initial solution

Circuit	Number of detected faults			# misclassifications		% misclassification	
	Spice3 / Init.-sol.	TBSA	1-iter.	TBSA	1-iter.	TBSA	1-iter.
leapfrog	656	656	591	0	99	0.00%	15.1%
continous	201	201	198	0	3	0.00%	1.49%
astabl	18	18	19	0	1	0.00%	5.56%
bias	94	94	98	0	4	0.00%	4.26%
latch	32	34	34	2	12	6.25%	37.5%
vreg	38	41	36	3	6	7.89%	15.8%
ab_integ	372	372	367	0	13	0.00%	3.49%
ab_opamp	82	83	86	1	22	1.22%	26.8%
gm6	90	90	90	0	0	0.00%	0.00%
schmitslow	75	75	70	0	5	0.00%	6.67%
gm2	58	58	58	0	0	0.00%	0.00%
gm3	58	58	58	0	0	0.00%	0.00%
Average	148	148	142	0.50	13.7	1.28%	9.72%

For some circuits, performing a single iteration causes a large number of fault misclassifications. Hence, this method cannot be used with high confidence to classify the faults for most circuits. On the other side, *TBSA* method shows a very low risk of fault misclassification, with no error for 9 of the 12 benchmark circuits and an average of 1.28 % errors for all the circuits. Even though *TBSA* requires about two times (2.14) more iterations than the single iteration method, the average percentage of parametric fault misclassification is reduced by a factor of 7.59, from 9.72 % for *1-iter.* to 1.28 % for *TBSA*.

The experimental results for catastrophic faults are presented in Table 4.5. One can notice that the reduction of the average number of iterations achieved by starting from an initial

solution (factor of 4.13) is not as important as for parametric faults (16.2). This is due to the fact that catastrophic faults correspond to structural modifications on the circuit that often result in a completely different behavior than the original fault-free circuit. Thus the initial solution used is often inaccurate and a large number of iterations are required before convergence. *TBSA* allows reducing significantly the number of iterations by a factor of 16.0 over *Spice3* and a factor of 3.87 (16.0/4.13) over *Init.-sol.* Although no catastrophic fault misclassification occurs for 7 of the 12 benchmark circuits, the percentage of misclassification reaches 3.87 % in average, which is larger than for parametric faults. This difference in the percentage of misclassification for parametric and catastrophic faults can be explained by the fact that starting from an inaccurate initial solution may introduce a large *WorstRelVar* value which overestimates the accuracy at the first iterations [19]. The exact solution is then incorrectly bounded and the iterations are stopped prematurely. The consequence is that *TBSA* reliability depends on the initial solution used. Performing a single iteration gives similar results for catastrophic and parametric faults with 11.5 % misclassification. When compared to *1-iter.* method, *TBSA* requires 3.85 times more iterations, but reduces the average percentage of catastrophic fault misclassification by a factor of 3. Although its reliability is not as good as for parametric faults, *TBSA* requires about 4 times less iterations than *Init.-sol.* method while causing 3 times less misclassifications than *1-iter.* method.

Table 4.5 : Iteration gain and fault detection for catastrophic faults reusing fault-free solution as initial solution

Circuit	Iteration gain			Number of detected faults			% misclassification	
	Init.-sol.	TBSA	1-iter.	Spice3 / Init.-sol.	TBSA	1-iter.	TBSA	1-iter.
leapfrog	3.02	6.35	37.4	365	365	334	0.00%	8.49%
continous	3.11	22.9	114	127	131	108	4.72%	16.5%
astabl	1.91	3.58	9.30	16	16	16	0.00%	0.00%
bias	3.66	13.1	43.6	91	91	94	0.00%	3.30%
latch	2.37	7.51	22.6	75	82	95	17.3%	34.7%
vreg	5.30	71.4	175	121	103	72	17.4%	43.8%
ab_integ	1.60	7.37	26.3	218	228	203	4.59%	13.3%
ab_opamp	14.0	79.7	259	120	123	106	2.48%	18.2%
gm6	1.90	4.79	10.4	31	31	31	0.00%	0.00%
schmitslow	0.98	2.28	17.4	42	42	42	0.00%	0.00%
gm2	2.40	3.45	11.0	34	34	34	0.00%	0.00%
gm3	1.80	3.11	12.8	45	45	45	0.00%	0.00%
Average	4.13	16.0	61.6	107	108	98.3	3.87%	11.5%

For *TBSA* method, Table 4.6 presents the speedup achieved by using a linear interpolation as described in Section 4.4.2 to generate the initial solution of the faulty circuits instead of reusing the fault-free solution. Only parametric faults are considered since linear interpolation is not applicable to catastrophic faults. Columns 2 and 3 report the average number of iterations required for each initial solution, while the speedup achieved by linear interpolation is reported in column 4. With using linear interpolation, the average number of iterations is only 1.54 per faulty circuit, for a speedup of 38.3 % over using the fault-free circuit solution and a total gain of 369 % (5.69/1.54) over *Init.-sol.* method. Moreover, no fault misclassification occurs. This shows that linear interpolation is very suitable for *TBSA* since both the number of iterations and the number of fault misclassifications are reduced.

Table 4.6 : Speedup achieved by linear interpolation for parametric faults using *TBSA* method

Circuit	# iterations		Speedup (%)
	Fault-free	Lin. inter.	
leapfrog	2.41	1.57	53.5
continous	2.91	1.97	47.7
astabl	1.57	1.19	31.9
bias	1.74	1.27	37.0
latch	1.94	1.39	39.6
vreg	1.32	1.07	23.4
ab_integ	1.81	1.26	43.7
ab_opamp	2.17	1.46	48.6
gm6	1.97	1.99	-1.0
schmitslow	3.92	2.51	56.2
gm2	2.41	1.69	42.6
gm3	1.55	1.14	36.0
Average	2.14	1.54	38.3

The simulation results show a significant reduction of the number of NR iterations required to perform DC fault simulation, at the cost of a low risk of fault misclassification. However, if *TBSA* is exploited into a test generation methodology, this error is much more negligible. Indeed, fault simulation tools are generally used in the inner loop of a testing algorithm which generates the optimal set of stimuli to detect all listed faults. Since each fault is simulated under a certain number of input stimuli, the risk that a single misclassification can be retrieved in the final optimal stimuli set is reduced. In other words, if a fault is misclassified for a set of stimuli, it may be correctly classified for the next set of stimuli. In order to avoid fault masking, many measured outputs are generally considered [26]. Therefore, a misclassification which occurs at one output may be eliminated if the fault is correctly classified at the other outputs.

The simulation results obtained by *TBSA* for catastrophic faults, parametric faults using fault-free solution and parametric faults using linear interpolation clearly demonstrate that the closeness of the initial solution is determinant for the performances of *TBSA*

method. For example, Table 4.6 shows that the use of linear interpolation has reduced by 38 % the CPU time for parametric faults, while the number of fault misclassifications has been reduced from 1.3 to 0.0 %. In the presented results, the initial solutions are generated by straightforward methods. However if, for a given circuit topology and injected fault, the faulty circuit solution can be predicted more accurately, the performances of *TBSA* in terms of CPU time and fault classification correctness will be improved. This suggests that some existing methods intended for linear analog circuits such as in [11], or dedicated to specific circuit topologies like in [14], may be reused to improve the initial solutions of faulty circuits and consequently improve *TBSA* performances.

4.6 Conclusion

We proposed a new simulation method to accelerate the analog DC fault simulation, allowing a rapid dropping of the easy-to-detect faults in DC simulation and reserving costly simulations to hard-to-detect faults. The proposed *TBSA* method starts from an initial solution and stops the NR iterations once a sufficient accuracy is obtained to correctly classify the fault. This heuristic has been validated on 12 benchmark circuits using a fault simulation algorithm that considers statistical process variations. *TBSA* is compared to other methods which reuse the fault-free solution as initial solution for the simulation of the faulty circuits. For parametric fault simulation, we showed that *TBSA* reduces the average number of NR iterations by a factor of 2.66 comparing to simulation until convergence, while causing only 1.28 % fault misclassifications. When compared to the single iteration method, *TBSA* is 2.14 times slower, but it reduces the average percentage of fault misclassifications by a factor of 7.59. For catastrophic fault simulation, *TBSA* reduces the number of iterations by a factor of 4, at the cost of less then 4 % fault misclassifications. We have also shown that any method which improves the initial solutions used by faulty circuits, such as linear interpolation technique for parametric faults, is likely to improve *TBSA* performances by reducing both the number

of NR iterations and the risk of fault misclassification. Hence, the CPU time required by a given test methodology can be significantly reduced, without noticeably degrading the results. *TBSA* method is currently studied to be integrated in an analog simulator that will be optimized for multiple simulations-based applications.

4.7 References

[1] G.G.E. Gielen, R.A. Ruterbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. of the IEEE*, vol. 88, no. 12, pp. 1825-1854, Dec. 2000.

[2] L.S. Milor, "A tutorial introduction to research on analog and mixed-signal circuit testing," *IEEE Trans. on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 45, no. 10, pp. 1389-1407, Oct. 1998.

[3] R.J.A. Harvey, A.M.D. Richardson, E.M.J.G. Bruls, K. Baker, "Analogue fault simulation based on layout dependent fault models," *Proc. of Int. Test Conf.*, pp. 641-649, Oct. 1994.

[4] J. Hou, A. Chatterjee, "Concurrent transient fault simulation for analog circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1385-1398, Oct. 2003.

[5] *STAR-HSPICE Quick Reference Guide 2001.4*, Avanti Co., Fremont, California, U.S.A., Dec. 2001.

[6] *SPECTRE Circuit Simulator User Guide, Product version 5.0*, Cadence Design Systems inc., San Jose, California, U.S.A., June 2003.

[7] M.W. Tian, C.-J.R. Shi, "Efficient DC fault simulation of nonlinear analog circuits," *Proc. of Design, Automation and Test in Europe*, pp. 899-904, Feb. 1998.

[8] S.J. Spinks, C.D. Chalk, I.M. Bell, M. Zwolinski, "Generation and verification of tests for analog circuits subject to process parameter deviations," *Journal of Electronic Testing: Theory and Applications*, vol. 20, no. 1, pp. 11-23, 2004.

[9] N.B. Hamida, K. Saab, D. Marche, B. Kaminska, G. Quesnel, "LIMSoft: automated tool for design and test integration of analog circuits," *Proc. of Int. Test Conf.*, pp. 571-580, Oct. 1996.

[10] P.N. Variyam, A. Chatterjee, "FLYER: fast fault simulation of linear analog circuits using polynomial waveform and perturbed state representation," *Proc. of 10th Int. Conf. on VLSI Design*, pp. 408-412, Jan. 1997.

[11] J.S. Augusto, C.F.B. Almeida, "Circuit equations for fast fault simulation and diagnosis of linear circuits," *IEEE Int. Conf. on Electronics, Circuits and Systems*, vol. 1, pp. 125-129, Sept. 1998.

[12] D.M.W. Leenaerts, J.V. Spaandonk, "DC testing of analog integrated circuits with piecewise linear approximation and interval analysis," *Proc. of Int. Symposium on Circuits and Systems*, pp. 1337-1340, May 1993.

[13] M. Zwolinski, C. Chalk, B.R. Wilkins, "Analogue fault modeling and simulation for supply current monitoring," *Proc. of European Design and Test Conf.*, pp. 547-552, March 1996.

[14] P.R. Wilson, Y. Kilic, J.N. Ross, M. Zwolinski, A.D. Brown, "Behavioural modeling of operational amplifier faults using analog hardware description languages," *Proc. of 5th IEEE Int. Workshop on Behavioral Modeling and Simulation*, pp. 106-112, Oct. 2001.

[15] M.W. Tian, C.-J.R. Shi, "Nonlinear analog DC fault simulation by one-step relaxation," *Proc. of 16th IEEE VLSI Test Symposium*, pp. 126-131, April 1998.

[16] P.N. Variyam, J. Hou, A. Chatterjee, "Test generation for analog circuits using partial numerical simulation," *Proc. of 12th Int. Conf. on VLSI Design*, pp. 597-602, Jan. 1999.

[17] J. Vlach, K. Singhal, *Computer Methods for Circuit Analysis and Design*, Van Nostrand Reinhold Company, U.S.A., 1983.

[18] T. Quarles, A.R. Newton, D.O. Pederson, A.S. Vincentelli, *SPICE3F User's Manual*, University of California at Berkeley, California, U.S.A., May 1993.

[19] M. Morneau, A. Khouas, "Analysis of DC simulation convergence of nonlinear analog circuits with initial solution," *IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 708-712, May 2005.

[20] A. Khouas, A. Derieux, "Fault simulation for analog circuits under parameter variations," *Journal of Electronic Testing: Theory and Applications*, vol. 16, no. 3, pp. 269-278, 2000.

[21] R.C. Melville, L.Trajkovic, S.-C. Fang, L.T. Watson, "Artificial parameter homotopy methods for the DC operating point problem," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, June 1993.

[22] M. Sachdev, B. Atzema, "Industrial relevance of analog IFA: a fact or a fiction," *Proc. of Int. Test Conf.*, pp. 61-70, Oct. 1995.

[23] P. Caunegre, C. Abraham, "Achieving simulation-based test program verification and fault simulation capabilities for mixed-signal systems," *Proc. of European Design and Test Conf.*, pp. 469-477, March 1995.

[24] B. Kaminska, K. Arabi, I. Bell, P. Goteti, J. L. Huertas, B. Kim, A. Rueda, M. Soma, "Analog and mixed-signal benchmark circuits – first release," *Proc. of Int. Test Conf.*, pp. 183-190, Nov. 1997.

[25] "Circuitsim90," *1990 Circuit Simulation and Modeling Workshop at MCNC.*

[26] S.D. Huynh, S.Kim, M. Soma, "Automatic analog test signal generation using multifrequency analysis," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 5, pp. 565-576, May 1999.

CHAPITRE 5

Article - MultiSPICE: Using an Optimized SPICE Simulator in the Inner-Loop of Fault Simulation, Circuit Sizing and Monte Carlo Applications

Michel Morneau, Abdelhakim Khouas
Electrical engineering department, École Polytechnique de Montréal

Abstract: Applications such as analog fault simulation, analog circuit sizing and Monte Carlo analysis tools invoke a large number of circuit simulations with slight modifications on the circuit components. This paper proposes a combination of methods to accelerate such applications which require multiple SPICE simulations. Proposed techniques include: communications between application and simulator through socket connection, efficient injection of modifications into nominal circuit, use of accurate initial DC solution approximations and methods allowing CPU time / accuracy tradeoff in DC simulation. Those techniques are implemented in the *MultiSPICE* analog circuit simulator, which is therefore optimized for multiple simulations. The three previously mentioned applications have been developed to exploit efficiently *MultiSPICE* as the encapsulated SPICE simulator. Like any SPICE-like simulator, *MultiSPICE* is independent of the calling applications. Experimental results on a set of benchmark circuits show that *MultiSPICE* is 85 times faster than *SPICE3f5* in DC simulation with the same simulation accuracy. For applications that can afford CPU time / simulation accuracy tradeoff, *MultiSPICE* is 125 times faster than *SPICE3f5*.

Index Terms: Analog circuit simulation, SPICE, Newton-Raphson method, Analog fault simulation, Analog synthesis, Analog circuit sizing, Monte Carlo analysis.

5.1 Introduction

The powerful CAD tools available today allow the digital circuit designers to achieve high productivity and performance. However, due to the high complexity of analog signals, few notable progresses have been accomplished in analog and mixed-signal domain over the years whereas most of the work remains done manually by experts. With the market of the mixed-signal circuits growing up to 20 % yearly, the analog part becomes the bottleneck of the total circuit design, even if it occupies only a small die area [1].

The most popular tool in analog domain is the circuit simulator. SPICE simulators, such as HSpice [2], Spectre [3] and Eldo [4], are very powerful tools that are able to predict the behavior of a given circuit under stimuli, taking advantage of accurate transistor models. The SPICE-like simulators use the Newton-Raphson (NR) iterative algorithm to solve the systems of nonlinear equations. Due to the growing size of analog circuits and the devices complexity, the simulation time remains large despite the use of faster computers. When a circuit has to be re-simulated for thousands of times with slight variations in the parameters, using SPICE-like simulators may require intractable CPU times.

5.1.1 Applications Requiring Multiple Simulations

Three major applications require many simulations of the same circuit with slight modifications: analog fault simulation (AFS), analog circuit sizing (ACS) and Monte Carlo (MC) analysis. Although other applications such as transistor-level static timing

analysis tools also require multiple simulations, this paper focuses on these three applications.

Analog fault simulation is a necessary step for the creation of a test set which verifies if the manufactured integrated circuit (IC) is altered by physical defects. In AFS tools such as ANTICS [5], each defect is modeled in a faulty circuit. The netlist file created for each faulty circuit is simulated by an inner-loop SPICE-like simulator to determine if the fault is detectable under the test set. Figure 5.1 illustrates the fault simulation flow. The CPU time may become considerable if a comprehensive test set is used. To reduce CPU time, concurrent simulators have been developed to take advantage of the similarities between fault-free and faulty circuits. In [6], the authors suggest sharing of the intermediate results among concurrently simulated circuits at each NR iteration. Other approaches propose specific ordering of the faults such that a good initial solution approximation is obtained [7], [8]. A single NR iteration method is proposed in [9] and [10] to approximate the simulation results, but there is no control over the accuracy achieved, which leads to unknown accuracy and higher risk of fault misclassification.

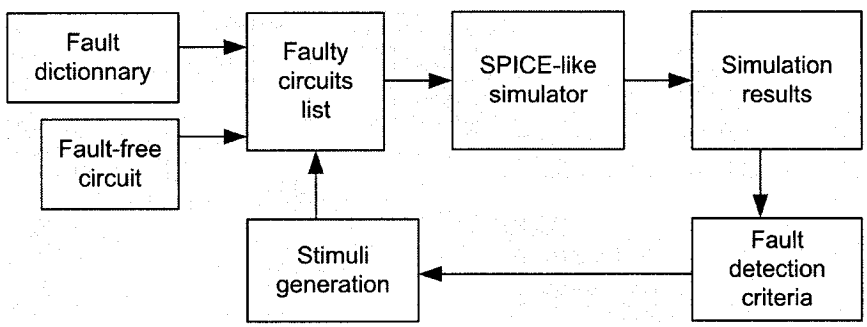


Figure 5.1 : Block diagram of standard AFS tools

ACS is the critical part of analog synthesis tools to find the optimal transistor dimensions to achieve design objectives and constraints. As mentioned in [11], the most accurate and trusted methods to perform analog circuit synthesis and circuit sizing, referred as simulation-based tools, require SPICE-level circuit simulation in the inner-loop of a

sizing algorithm. Candidate circuits are generated by the sizing algorithm and SPICE simulations produce the results required to evaluate their performances. More than 100 000 candidate circuits may be explored before obtaining the desired circuit performances for medium or large circuits [12]. To accelerate DC simulation, [13] reuses previous solutions as the initial solution approximation, whereas [14] predicts the starting solution from previous results. For these two techniques, a customized simulator has been developed. The use of an encapsulated SPICE-like simulator is suggested in ANACONDA [12] and AMIGO [15] and illustrated in Figure 5.2. This approach has the main advantage to hide the simulator implementation characteristics from the sizing algorithms, at the expense of larger CPU time.

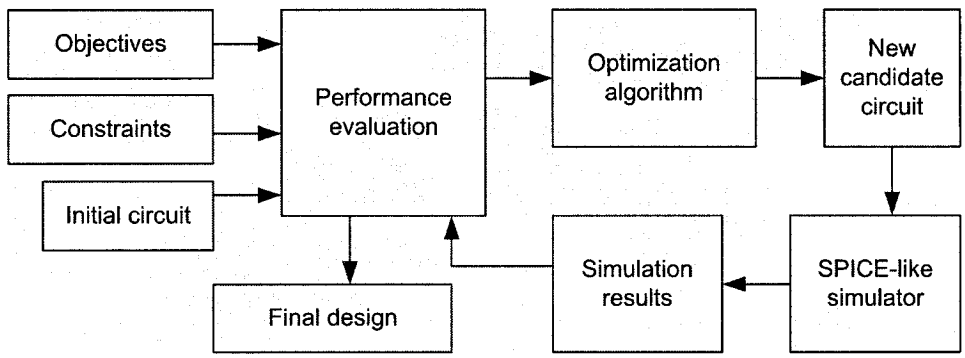


Figure 5.2 : Block diagram of simulation-based ACS tools

MC analysis is used to predict the effects of the manufacturing process fluctuations on the performances of the integrated circuits (IC). A certain number of circuit samples are generated according to the manufacturing process information. The simulation results of the circuit samples are analyzed to evaluate the robustness of the circuit. MC simulations are mainly used to improve the robustness of analog circuits in order to enhance quality and manufacturing yield and drive down component costs [16], [17]. Realistic analog fault simulation also requires MC simulation [18], [19]. The implementation of MC analysis using a SPICE-like simulator is shown in Figure 5.3. Although some simulators offer their own Monte Carlo function, the user must use the options provided by the simulator and hence has a limited control on the generated circuit samples and the

required simulation accuracy. On the other hand, the two steps linear approximation method [20] and FASTEST tool [21] have proved to accelerate the DC simulation of each sample circuit. However, such methods require the customization of a simulator.

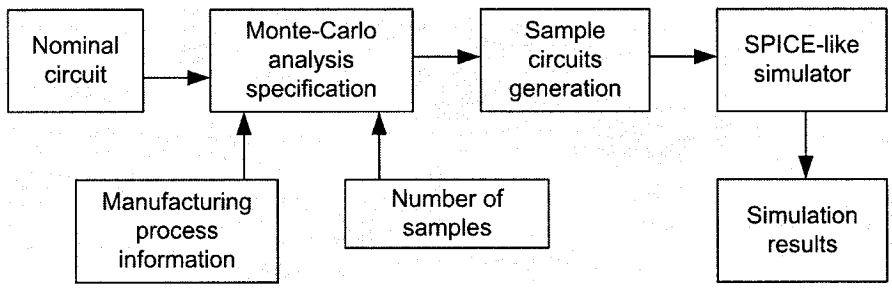


Figure 5.3 : Block diagram of MC analysis tools

The common point among the three mentioned applications is the multiple simulations of the same circuit, with slight modifications from the nominal circuit. Moreover, full simulation accuracy is not required as long as the results produced by each application are not altered. Indeed, the AFS application only requires classifying a given fault as detected or not under specified stimulus; an ACS tool needs to keep the promising circuits and reject the mediocre ones; and MC analysis is performed to ensure that the performances are not altered by the manufacturing process variations. Although customized simulators have proved to achieve the best CPU time performances, the use of an independent encapsulated simulator favors a better portability and separate efforts on both applications and simulation tools.

5.1.2 Contributions of this Work

As suggested in [12], the ideal simulator for multiple simulations is one which can be invoked once and, remaining live, can interpret quickly a stream of requests to modify circuit values, simulate and return the results. However, SPICE-like simulators are designed for interactive schematic-update-simulate operation for a single circuit, where overheads of a few seconds per simulation are considered negligible. Nevertheless, when

thousands of circuits are simulated, the addition of such overheads may be intractable. In this paper, the term "modified circuit" refers to any circuit derived from the original one and on which some parameter or structural modifications have been injected.

The objective of this paper is to propose a new SPICE-like simulator that significantly reduces the CPU time required by the multiple simulations. Although the proposed methods target the three previously mentioned applications, other applications with multiple simulations are also likely to take advantage of these methods. Four key ideas are proposed in this paper:

1) *Socket communications*: The communications between the application and the simulator are performed via socket connection using TCP protocol instead of using intermediate disk files.

2) *Efficient injection of modifications*: The modified circuits are described by injecting modifications on the nominal circuit instead of creating new netlist files.

3) *Accurate initial DC solution approximation*: The previous simulation results are reused to compute the initial solution approximation of each modified circuit, according to an equation specified by the calling application.

4) *Accuracy / speed tradeoff in DC simulation*: Three methods are proposed to end the NR iterations once the needed accuracy is achieved and therefore reduce the CPU time in DC simulation.

These improvements have been included in an analog circuit simulator named *MultiSPICE*. We also developed three applications which take advantage of *MultiSPICE* as the encapsulated analog simulator. The proposed AFS, ACS and MC applications feature a graphical user interface and fully exploit the functions offered by *MultiSPICE* to reduce the CPU time required by the repetitive simulations.

This paper is structured as follows: section 5.2 discusses the proposed methods. Their implementation in *MultiSPICE* simulator is exposed in section 5.3. Section 5.4 presents

AFS, ACS and MC analysis applications. Section 5.5 shows some experimental results. Concluding remarks follow in section 5.6.

5.2 Proposed SPICE Simulation Methods

In this section, we describe the proposed methods to reduce the CPU time required by applications which perform multiple simulations of similar analog circuits. These methods are summarized in four key ideas: socket communications, efficient injection of modifications, accurate initial DC solution approximation and accuracy / speed tradeoff in DC simulation.

5.2.1 Socket Communications

The use of a SPICE-like simulator in the inner loop of applications such as ANTICS [5] or ANACONDA [12] generally requires the following operations for each modified circuit:

1) A netlist file describing the modified circuit is created by the application.
2) The netlist file is parsed and the circuit structure is built in the simulator.
3) The simulation jobs are performed on the circuit.
4) The simulation results are saved in disk file by the simulator.
5) The results file is parsed by the application.

In these operations, at least two disk files are written and read for each modified circuit. Hard disk accesses are relatively long and are generally measured in milliseconds. On the other side, accesses in RAM can be measured in microseconds. Hence, the use of a direct communication link eliminates the need of multiple hard disk accesses to write / read the netlist and results files. A socket is a descriptor allowing two applications on the same or different machines to communicate in a transparent way via the chosen protocol. Transmission Control Protocol (TCP) enables a connection between two hosts and

exchange streams of data. The delivery of the data is guaranteed, in the same order as sent. This protocol is well suited for SPICE commands, which can be followed by any number of arguments. To reduce the time spent to transmit modified circuits to the simulator, we therefore replace the use of temporary files by a socket connection using TCP protocol between the application and the simulator.

In the proposed communication flow, *MultiSPICE* simulator remains alive for as long as required by the application, as illustrated in Figure 5.4. At first, the simulator reads the netlist file of the original circuit. *MultiSPICE* stays in an idle state and waits for instructions from the application. Then, the application sends the simulation jobs and options and the description of modified circuits to *MultiSPICE* via the socket connection. Once the simulation is completed on each circuit, the results are returned to the application using the same socket connection. *MultiSPICE* then returns in idle state until new instructions are received, and so on. As no intermediate file is used, the disk access time is completely eliminated.

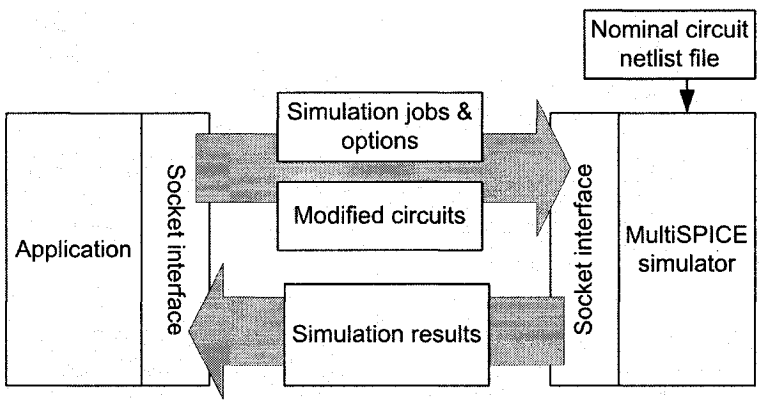


Figure 5.4 : Flowchart of applications using *MultiSPICE* simulator in an inner-loop

5.2.2 Efficient Injection of Modifications

When using SPICE-like simulators, each circuit must be described by a netlist file. However, in the target applications, the difference between the nominal circuit and each

of the modified circuits is generally small. Specifying the modifications to apply on the nominal circuit is almost always simpler than creating a new netlist file for each modified circuit. Indeed, a few lines can describe such modifications instead of a complete netlist in which each circuit component is described. Moreover, when a netlist file is read by the simulator, each circuit component is parsed, created and added into the new circuit description structure before beginning simulation. CPU time can be saved if the nominal circuit description is reused and each modification is directly injected in the nominal circuit, since the complete structure is not rebuilt. Concurrent fault simulators [6], [7], [8] take advantage of such circuit reutilization.

SPICE-like simulators are not suited to specify both parameter and structural modifications on a nominal circuit. Although component or model parameter values can be modified with the keyword "*.alter*", the nominal value is lost and the new value is reused for all future simulations unless a restore process is considered. Moreover, although a structural modification such as a short could be specified by adding a new resistor, this new component cannot be easily removed afterward.

MultiSPICE simulator offers new instructions allowing the addition of both parameter and structural modifications to the nominal circuit. A modification is only applied on the current modified circuit, not on the next ones. In other words, once each modified circuit is simulated, the nominal circuit is restored before the injection of the modifications associated to the next modified circuit. The following modifications can be injected into the nominal circuit:

Component / model parameter value: Alter the specified component or model parameter to a new value.

Short between nodes: Add a resistive and/or capacitive short between two specified circuit nodes.

Short between component terminals: Add a resistive and/or capacitive short between two terminals of the specified circuit component.

Nonlinear component open: Force the specified transistor or diode to remain in the cut-off region.

Those simple types of modifications, and combinations among them, allow describing most of the possible structural faults, component size variations and parameter deviations.

5.2.3 Accurate Initial DC Solution Approximation

In DC simulation, the NR iterative algorithm is used by SPICE-like simulators to solve the systems of nonlinear equations. The NR algorithm starts from an initial solution approximation x^0. The circuit components are linearized around this solution approximation according to the Modified Nodal Analysis (MNA) notation [22], which generates a system of linear equations. The solution of this system is a new, hopefully better, approximation of the exact solution. A second iteration is performed from this approximation and so on, until the convergence is reached. Equation (5.1) represents the k^{th} NR iteration, where J is the Jacobian matrix (also called linearized conductances), x^{k-1} is the previous solution approximation, x^k is the new solution approximation and F is the minimization function. Both J and F are built during the component linearization phase from the equations included into the encapsulated component models.

$$J(x^{k-1})(x^k - x^{k-1}) = -F(x^{k-1}) \qquad (5.1)$$

The convergence is reached and the NR iterations are ended when the results of two consecutive iterations differ by less than a given tolerance for each node. The final solution approximation obtained at convergence is considered as the solution of the system of nonlinear equations and is returned by the simulator. As shown in equation (5.2), the tolerance tol_i^k for the node i at the iteration k depends on absolute (*abstol*) and

relative (*reltol*) tolerances and on the node's value at current (x_i^k) and previous (x_i^{k-1}) iterations. The convergence criterion used in *SPICE3f5* [23] is shown in (5.3).

$$tol_i^k = abstol + reltol \times \max\left(|\, x_i^k \,|, |\, x_i^{k-1} \,|\right) \qquad (5.2)$$

$$\text{if } |\, x_i^k - x_i^{k-1} \,| \le tol_i^k \;\; \forall\, i \;\; \rightarrow \text{convergence} \qquad (5.3)$$

Default values used in SPICE simulators [3], [23] are *reltol* = 10^{-3}, *abstol* = 10^{-6} V for voltages and *abstol* = 10^{-9} A for currents. The NR algorithm exhibits quadratic convergence when the solution approximation is close to the exact solution of the system of nonlinear equations [22]. Otherwise, no convergence assertion can be made and additional iterations are needed before reaching convergence. In this paper, the exact solution refers to the final solution returned by NR algorithm once usual convergence criterion is met, whereas the solution obtained at each iteration is referred as solution approximation. Since the CPU time is proportional to the number of NR iteration, minimizing the number of iterations proportionally reduces the CPU time.

SPICE-like simulators start the NR algorithm with an arbitrary initial solution approximation which is generally too far from the exact solution, so 10-100 iterations are usually required before DC convergence. As shown in [24], starting from an accurate initial solution approximation (AISA) avoids useless iterations and hence the number of iterations is reduced without altering the simulation results. Moreover, convergence problems can be greatly reduced [25], which improves simulation robustness. For applications which require multiple simulations, the AISA can be computed from the solution obtained for previous simulated circuits. For example, the simulation result of the nominal circuit can be reused as the AISA for all other circuits. The use of AISA in the context of multiple simulations is complex to manage with SPICE-like simulators because the nominal circuit solution must be saved in temporary file and then reloaded for each circuit which needs AISA. Additionally, a better AISA could be used, such as reusing the solution of the previous circuit instead of nominal circuit, or interpolating

AISA from two or more previous circuits' solutions, as proposed in [8]. However, such interpolation cannot be performed automatically in existing SPICE-like simulators.

When a modified circuit is specified to *MultiSPICE* simulator, an interpolation equation is provided to set the AISA of each DC point. This initialization equation is defined as a function of the solutions of previous circuits. The DC operating point is then started with the AISA computed by the provided interpolation equation. The main point is that the calling application can take advantage of its knowledge on the injected modifications to predict an AISA which is as close as possible to the exact solution. Hence, the NR algorithm is likely to converge rapidly.

AC and transient analyses are also accelerated since they require the DC operating point computation as their starting point. In the case of DC sweep analysis, a linear interpolation is performed to set the AISA of the current step from the initialization equation. As shown in equation (5.4), the AISA at current step s (x_s^0) is computed as the sum of the exact solution of previous step (x_{s-1}^0) and a correction term. The correction term is the difference between the value at the current ($x_{s,pred}$) and previous ($x_{s-1,pred}$) steps predicted by the initialization equation.

$$x_s^0 = x_{s-1} + \left(x_{s,pred} - x_{s-1,pred}\right) \qquad (5.4)$$

Subtracting current and previous steps predictions helps canceling the biases that systematically occur if the interpolation equation provided by the application is not enough accurate to generate initial solution approximations close to the exact solutions.

5.2.4 Accuracy/Speed Tradeoff in DC Simulation

As mentioned in the introduction of this paper, the target applications do not generally need full simulation accuracy as long as the returned results remain valid. In DC

simulation, the number of NR iterations can be reduced if the NR algorithm is stopped before convergence. Three previously published techniques to end NR algorithm before convergence have been included in *MultiSPICE* simulator as available options. They require an AISA as starting point of NR algorithm.

The first technique allows ending the NR algorithm after a predefined number of iterations. This method has been experimented in [10] for DC tests generation. The best speed/accuracy tradeoff for this method has been achieved with the number of iterations fixed to one, instead of any larger number. The use of a single iteration has also been used to predict rapidly the fault coverage of nonlinear analog circuits [9] and to approximate mean and standard deviation values in MC analysis [20]. The main disadvantage of this method is the absence of control over the accuracy achieved when the simulation is stopped. Although for most modified circuits the obtained solution is close to the exact solution, for others, the results are not enough accurate for the requirements of the application. For example, a faulty circuit may be misclassified by the AFS tool, or the performances of a candidate circuit may be overestimated during ACS. Nevertheless, this single iteration method can provide a good approximation of the exact solution in most cases, especially when the initial solution approximation is expected to be close to the exact solution. For example, this is usually the case in Monte Carlo simulation.

Since the default relative tolerance parameter (*reltol*) value is 10^{-3} (0.1 %) or less in SPICE-like simulators, the maximal error on each node at convergence is considered negligible compared to the accuracy of the transistor models. The second method, detailed in our previous works [24], proposes to relax *reltol* to the accuracy needed by the calling application. This method avoids performing the last iterations before convergence, which are considered useless for the required accuracy. The expected error on each circuit node is assumed to be less than the *reltol* value. Although the usual convergence criterion is used to end iterations, we cannot assume that the convergence is reached

when *reltol* is relaxed since the obtained solution is not exact. Indeed, the obtained solution is an approximation of the exact solution, with an expected (but not guaranteed) accuracy equals to the *reltol* value. Experiments in [24] show that the number of iterations can be reduced by 14 to 65 %, depending on circuit topology, goodness of AISA and required accuracy.

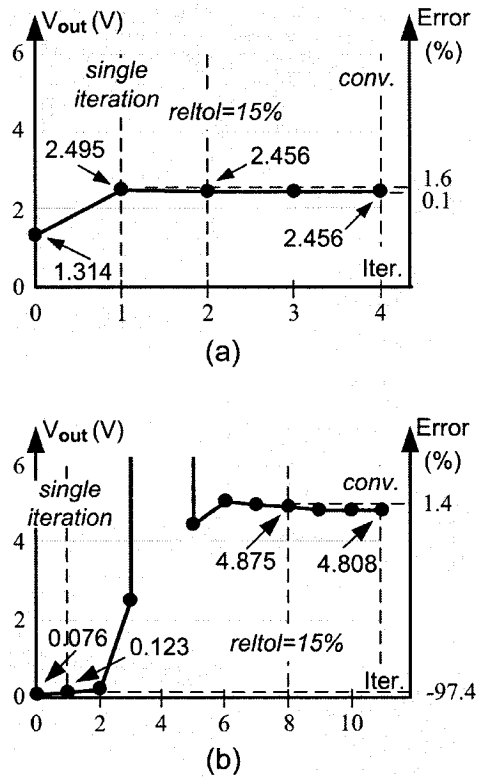


Figure 5.5 : Accuracy / number of iterations tradeoff achieved by simulation until convergence, single iteration and *reltol* relaxation methods for (a) accurate and (b) inaccurate initial solution approximation

Figure 5.5 illustrates the output voltage behavior during NR iterations until convergence (with *reltol* = 0.1%) and compares the single iteration with *reltol* relaxation methods. For each example, the output voltage obtained at each iteration is on the left axis whereas the error achieved by each method is shown on the right axis. The output voltage produced by each of the three compared methods is also pointed out. Let us assume that the needed

accuracy for one application is ±15 %. In Figure 5.5 (a), the provided AISA is accurate and only 4 iterations are needed before convergence. With *reltol* = 15 %, the criterion (2)-(3) ends the iterations after the 2nd iteration (with 50 % less iterations), whereas single iteration also produces good results with one iteration (75 % less iterations). However, in Figure 5.5 (b), the provided AISA is inaccurate and 11 iterations are required before convergence. With *reltol* = 15 %, the *reltol* relaxation method performs 8 iterations (27 % less iterations) and produces accurate results, whereas single iteration method shows an erroneous output voltage. Hence, *reltol* relaxation method offers a speed / accuracy tradeoff between simulation until convergence and single iteration methods.

The last proposed method to end NR algorithm before convergence has been presented by the authors in [26] for AFS. In this paper, we present the generalization of this method, which can easily be used for ACS or any other application that needs to compare the simulation results of modified circuits. Threshold-Based Simulation Accuracy (*TBSA*) method ends the iterations when the solution approximation x^k at k^{th} iteration is enough accurate to perform the comparison. This method uses an approximated value of the accuracy achieved at each iteration, which is given by the *WorstRelVar* metric. This metric is the largest relative variation affecting a circuit node between two consecutive iterations k-1 and k, as shown in equation (5.5):

$$\mathbf{WorstRelVar}^k = \max_{\forall i}\left(\frac{|\, x_i^k - x_i^{k-1}\,|}{\max\left(|\, x_i^k\,|, |\, x_i^{k-1}\,|, \mathbf{abstol}\right)}\right) \qquad (5.5)$$

Let V be the DC output voltage of the modified circuit to be computed by simulation. After iteration k, the minimal (V_{min}^k) and maximal (V_{max}^k) expected values of the output voltage V are computed from the approximated voltage V^k at iteration k, according to equation (5.6):

$$\begin{aligned} V_{\min}^{\ k} &= \left(1 - \mathit{WorstRelVar}^k\right)V^k \\ V_{\max}^{\ k} &= \left(1 + \mathit{WorstRelVar}^k\right)V^k \end{aligned} \qquad (5.6)$$

If we assume that the application needs the simulation results to compare the output voltage V to an interval defined by two threshold values $V_{th,min}$ and $V_{th,max}$, the simulator can use these two threshold values to end the iterations before convergence. The minimal V_{min}^{k} and maximal V_{max}^{k} expected output values obtained at each iteration k are compared to the threshold values and one of the three following cases occurs:

1) Both V_{min}^{k} and V_{max}^{k} are inside the interval $[V_{th,min}, V_{th,max}]$. We consider that the response V lies in this interval and the iterations are immediately ended.

2) Both V_{min}^{k} and V_{max}^{k} are either lower or higher than the interval $[V_{th,min}, V_{th,max}]$. Therefore the response V is assumed to be outside the interval and the iterations are ended.

3) V_{min}^{k} and V_{max}^{k} are on each side of the threshold $V_{th,min}$ or $V_{th,max}$. In this case, the iterations are continued to improve the solution accuracy.

In AFS, the fault is automatically considered detected if both V_{min}^{k} and V_{max}^{k} are either lower or higher than the interval while it is not detected if V_{min}^{k} and V_{max}^{k} are inside the interval. In ACS, one of the threshold values may be set to + or - infinity. Hence a candidate circuit is automatically classified as rejected if its output range $[V_{min}^{k}, V_{max}^{k}]$ is higher or lower than the remaining threshold. The example of Figure 5.6 illustrates the graphical interpretation of *TBSA*. At the first iteration, V_{min}^{1} is below $V_{th,min}$, whereas V_{max}^{1} is above $V_{th,min}$. Hence, from these results, V could either be above or below $V_{th,min}$, so the iterations are continued. After the second iteration, V_{min}^{2} and V_{max}^{2} remain on each side of $V_{th,min}$ and another iteration is needed. After the third iteration, both V_{min}^{3} and V_{max}^{3} are below the threshold $V_{th,min}$, so the output value V is expected to be below $V_{th,min}$. Three iterations are sufficient to correctly compare the modified circuit output value to the considered threshold interval.

The modifications which cause large deviations on the circuit behavior are usually those that require a large number of iterations when the nominal circuit provides the AISA. For such modified circuits, *TBSA* ends the simulation even if much more iterations are

required to obtain a more accurate solution, saving substantial CPU time. Both speed and accuracy of *TBSA* method are enhanced when the provided AISA is close to the exact solution [26].

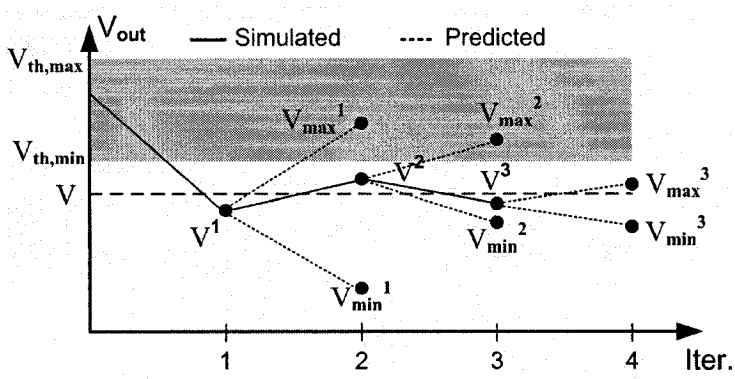


Figure 5.6 : Simulation example using *TBSA* method

Those four simulation methods are available in *MultiSPICE*: until convergence, fixed number of iterations, relaxed *reltol* value and *TBSA*. Depending on the requirements of the application, the appropriate method can be selected to perform DC simulation on the modified circuits.

5.3 Implementation

This section describes the implementation of the methods previously discussed into the *SPICE3f5* [23] analog circuit simulator. The resulting simulator is *MultiSPICE*.

5.3.1 Overview of *MultiSPICE* Simulator

MultiSPICE reuses the same device models equations and NR algorithm implementation as *SPICE3f5*. The supported analysis types are DC operating point (.op), DC sweep (.dc), AC (.ac) and transient (.tran) simulations. The available options are the same as *SPICE3f5*, with the addition of *MultiSPICE*-specific options.

MultiSPICE offers a new interactive command interface, allowing the simulator remaining active for as long as required by the calling application. Figure 5.7 shows the sequence of steps performed by an application that uses *MultiSPICE* as the encapsulated simulator. The only netlist file parsed by *MultiSPICE* is the original circuit. The list of circuit nodes, components and parameter values can be requested by the application afterwards. The modified circuits are specified by adding modifications to the original circuit topology or device parameters. The simulation jobs and options are specified as in *SPICE3f5* simulator but the instructions are directly sent via socket connection. The simulation results are returned to the application once available.

1 Begin the application

2 Start MultiSPICE simulator

3 Establish connection with MultiSPICE

4 Send a *.source* command to specify netlist file

5 Request for the nominal circuit information

Figure 5.7 : Steps performed by an application for using *MultiSPICE* simulator in an inner-loop

5.3.2 Data Structure

To accelerate future simulations, the information on each modified circuit and the simulation results are stored in a dedicated data structure. Figure 5.8 presents the top-down hierarchy of the data structure.

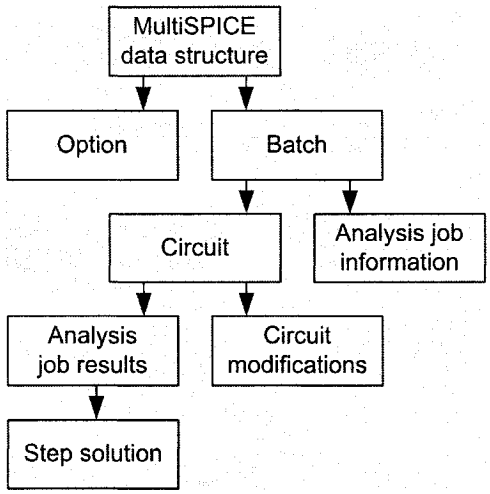


Figure 5.8 : Data structure used to store circuits description and simulation results

The list of modified circuits sent simultaneously to the simulator in the same communication is called a batch. Therefore, a batch structure stores the description of each analysis job and the information on each owned circuit. The same analysis jobs (op, dc, ac, tran) and simulation options are shared by all the circuits in a batch. For an existing batch, *MultiSPICE* does not allow adding or removing circuits in upcoming commands, but additional simulation jobs can be performed on the batch. The nominal circuit remains in a separate batch. Any number of batches can be defined.

A modified circuit is described by the modifications to apply on the nominal circuit, as mentioned in subsection 5.2.2. Once a simulation job is completed on a circuit, the simulation results are stored in the circuit data structure and returned to the application once the execution of the entire batch of circuits is completed. In order to free memory space, the application can at any time request from *MultiSPICE* the deletion of useless circuit description or simulation results.

This data structure allows storing all the relevant information suitable for the acceleration of future simulations without accessing disk files or repeatedly transfer the same information. Thus, by managing efficiently the information and simulation results on

each modified circuit, the calling application can achieve a better memory / CPU time tradeoff over traditional simulators.

5.3.3 Example of Circuit Simulation using *MultiSPICE*

Figure 5.9 presents an example of batch instructions for modified circuits. Those instructions are sent by the application to *MultiSPICE* via the socket interface using TCP protocol.

```
1     .option outnode n2, n3
2     .batch 1
3     .tran 1e-3 1e-5
4     .dc v1 -2 3 1
5     .circuit 1 $0 store
6     .cparam m1 w 5e-6
7     .cparam m2 w 2e-6
8     .circuit 2 ($0+$1)/2 store
9     .mparam mos1 tox 1e-9
10    .circuit 3 2*$2 store
11    .short n1 n2 100
12    .circuit 4 3 flush
13    .off m1
14    .short m1 g s 10e6 10e-12
15    .short m1 d s 10e6 10e-12
16    .short m1 d g 10e6 10e-12
17    .end
```

Figure 5.9 : Example of batch instructions for modified circuits

On the first line, nodes *n2* and *n3* are set as the outputs, so the simulation results associated with these nodes will be returned to the calling application. The new batch is specified by its unique ID on line 2. Transient and DC sweep analyses are to be performed (lines 3-4). Lines 5-7 describe the modified circuit with ID #1. The nominal circuit (ID #0) solution is used as AISA, and the "*store*" keyword means that simulation results remain stored in the data structure, so they can be reused to compute AISA of

other modified circuits. Modified circuit #1 has *m1 width* = 5 µm and *m2 width* = 2 µm instead of nominal values. Modified circuit #2 (lines 8-9) has model parameter *tox* = 1n, uses the average between nominal circuit and circuit #1 as AISA. Circuit #3 (lines 10-11) has a 100 Ω short between nodes *n1* and *n2*, its AISA is two times the circuit 2 solution. Modified circuit #4 (lines 12-16) has *m1* forced in cut-off region, with 10 MΩ resistances and 10 pF capacitances between its terminals, it reuses circuit #3 solution as AISA and the simulation results are deleted. The *.end* keyword on line 17 indicates the end of transmission and *MultiSPICE* executes the simulation jobs on the modified circuits.

Once the simulation is completed, the results are immediately returned by *MultiSPICE* on the communication link. Sample results are shown in Figure 5.10.

```
1    .batch 1
2    .anal tran 1e-3 1e-5
3    .circuit 1
4    .node n2 0.5 0.51 0.53 ...
5    .node n3 2.5 2.51 2.53 ...
6    .circuit 2
7    .node n2 0.45 0.47 0.51 ...
8    .node n3 0.45 0.47 0.51 ...
(...)
9    .dc v1 -2 3 1
10   .circuit 1
11   .node n2 0.4 0.51 0.63 0.75 0.89 1.09
12   .node n3 2.3 2.56 2.79 2.99 3.28 3.55
(...)
14   .end
```

Figure 5.10 : Example of results returned by *MultiSPICE* after batch execution

The batch is first specified by its unique ID on line 1. On lines 3-5, transient simulation results are returned for each output of modified circuit #1, which are *n2* and *n3*. This is repeated for each modified circuit. Then, DC sweep simulation results are returned (lines

9-12) for each circuit and each output node. The *.end* keyword indicates the end of the communication.

5.4 Applications

We developed three applications that perform multiple simulations: AFS, ACS, and MC analysis tools. This section describes these applications which take advantage of *MultiSPICE* as the inner-loop circuit simulator.

5.4.1 Analog Fault Simulation Tool

Our AFS application let the user specify the fault list on a graphical user interface. Parametric faults can be injected on both component and model parameters. Resistive and/or capacitive shorts can be injected between terminals of specified components or between two specified circuit nodes. An open fault is injected in nonlinear components by turning off the component. Opens in resistors are injected by altering their resistance to a high value whereas opens in capacitors are injected by altering their capacitance to a low value.

Figure 5.11 shows an example of fault list used to generate faulty circuits. The back window displays the current fault list. The type of fault is shown in the first column, whereas affected components/nodes/parameters/terminals are listed in the second column. Faulty values for parametric faults and short resistances for short faults are displayed in column 3, whereas column 4 shows the number of faulty circuits associated with each fault definition. If more than one fault is injected on each faulty circuit, the 5^{th} column would indicate yes. Many faulty samples can also be generated for each fault if the user wants to take into account process variations. The front window shows the definition of faulty component parameter. Such fault definition generates a certain number of faulty circuits with a single fault into each circuit. The number of faulty

circuits is equal to the number of instances of the specified component in the circuit, multiplied by the number of faulty values. In this example, the fault affects the parameter *c* (capacitance) of all capacitors with model *C*. The faulty values are 6 and 7 times the nominal value of each capacitor. The front window bottom list is optional and allows specifying additional faults to inject in the generated faulty circuits for parallel fault simulation. An additional fault can only affect one component in the circuit and is duplicated in each faulty circuit. In the example, an additional *vin dc* = 2 V fault is added into each of the previously generated faulty circuits, for a total of two faults for each faulty circuit. Three methods are offered to set the AISA of each faulty circuit. In the first method, we reuse the exact solution of the nominal circuit. In the second one, we order the faulty circuits according to the faulty value or short resistance and we reuse the solution of the previous faulty circuit. The third method is only available for parametric faults and performs a linear interpolation over faulty values according to equation (5.7) and similar to [8]. The AISA x_j^0 for the faulty circuit j with faulty parameter value P_j is the linear projection of the two previous faulty circuits j-1 and j-2, scaled by the difference of values for the parameter.

$$x_1^{\,0} = x_0$$
$$x_j^{\,0} = x_{j-1} + (x_{j-1} - x_{j-2})\frac{P_j / P_{j-1}}{P_{j-1} / P_{j-2}} \quad (j \geq 2) \qquad (5.7)$$

Once the fault list is built, the faulty circuits are generated. For the simulation, a batch of faulty circuits is sent to *MultiSPICE* simulator with the considered outputs and analysis jobs. A fault is considered detected if its output differs from the nominal circuit output by more than a given threshold. For the fault-free and faulty circuit comparison, three different fault detection thresholds are available:

- Absolute deviation value (e.g. ±50 mV).
- Percentage of the fault-free output value (e.g. ±5 %).
- Factor of the fault-free distribution (e.g. ±3*standard deviation value of nominal circuit).

The third detection threshold type requires the fault-free circuit mean and standard deviation values, which can be computed by Monte Carlo application.

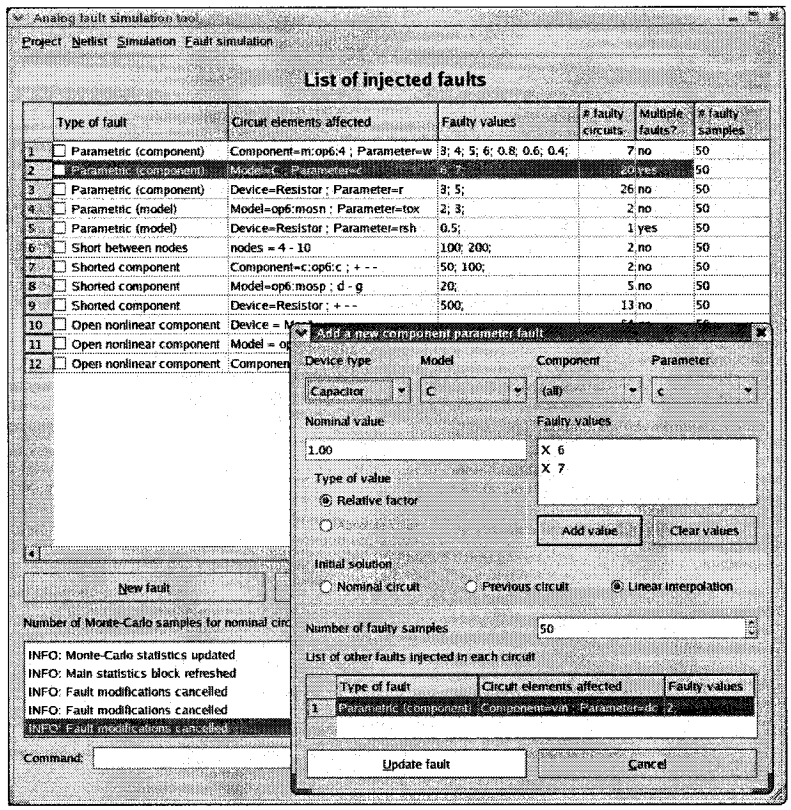


Figure 5.11 : Example of fault list and parametric fault specification

5.4.2 Analog Circuit Sizing Tool

The ACS tool presented here is a basic tool allowing randomly modifying some circuit parameters and ordering them according to custom objectives and constraints. The user first selects the modifications to apply to the varying circuit parameters. Fixed or random values can be specified for each of the considered parameters. Uniform, Gaussian and corner (extreme values) distributions are available to generate each random value. Each generated candidate circuit is affected by modifications on each of the specified varying parameters. Therefore, simulations allow detecting which modifications are likely to

produce the best circuit performances. For the simulation, the original circuit provides the AISA of each candidate circuit. Once simulated, the candidate circuits are ordered according to how well they meet the circuit requirements. Such requirements are described by the following elements:

1) Hard constraint: If one hard constraint is not met, the candidate circuit is automatically rejected.

2) Soft constraint: When a circuit does not meet a soft constraint, the tool applies a penalty proportional to how much this constraint is broken.

3) Objective: For each simulation point, the objective can be minimizing or maximizing the output value.

Each soft constraint and objective is defined by its weight value. The combination of soft constraint and objective generates a weighted performance value for each simulation point. Therefore, a single performance value is obtained for each circuit candidate, which takes into account all simulation points. Figure 5.12 illustrates the results summary obtained from candidate circuits. For each simulation point, the performance value has been computed. Some circuits are rejected since they broke a hard constraint. The other candidates are ordered according to their performance value.

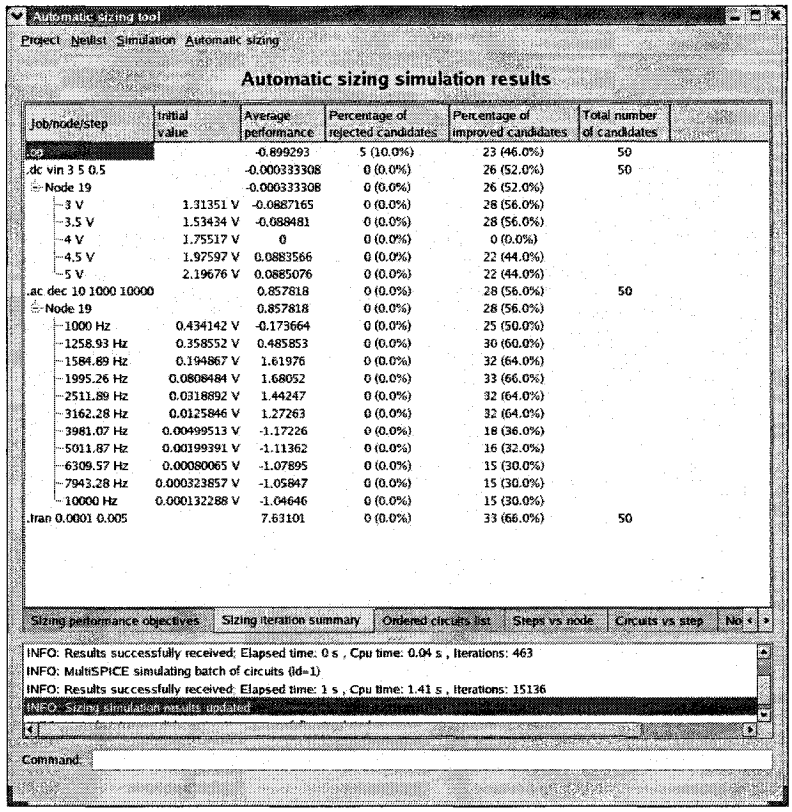


Figure 5.12 : Candidate circuits ordered by performance value in circuit sizing tool

This simple ACS tool requires from the user to specify manually the variations on the circuit parameters for the new candidate circuits, according to the most promising simulated candidates. However, this behavior is considered sufficient to validate the performances (speed and accuracy) of *MultiSPICE* simulator in the inner-loop of any complete ACS or analog synthesis application.

5.4.3 Monte Carlo Application

The proposed MC application allows the user to set manually and easily the process deviations affecting circuit components via graphical user interface. Component and model parameters, as defined in *SPICE3f5* [23], can be affected by random deviations.

Both relative and absolute deviations can be set on each parameter. Uniform, Gaussian and corner distributions are available to generate each random value.

The application supports both process and mismatch variations. Process variations affect a certain number of components in the circuit in the same way, whereas mismatch variations affect a single component. Components affected by the same parameter deviation can be grouped in the same statistics block. Moreover, each component into a statistics block can also have additional individual random variations which are added to the global deviation of the statistics block. Each statistics block can contain any number of sub-blocks, allowing the definition of a custom hierarchy.

Figure 5.13 shows the GUI offered by MC application to set random parameter deviations for a given statistics block. The parameter and block information and the global random deviations affecting all child blocks and components are shown in upper part of the window. The upper table displays child blocks with their local parameter deviations whereas the lower table displays component instances, with netlist (default) parameter values and their local parameter deviations. In this example, 3 % random deviations are applied to each *Mos1 width* parameter with uniform distribution. *Width* parameter of each component included in sub-blocks is affected by an additional 2 % random deviation with Gaussian distribution. *Width* parameter of each component in current block is affected by an additional 1 % random deviation with uniform distribution and ±10 μm absolute random deviation.

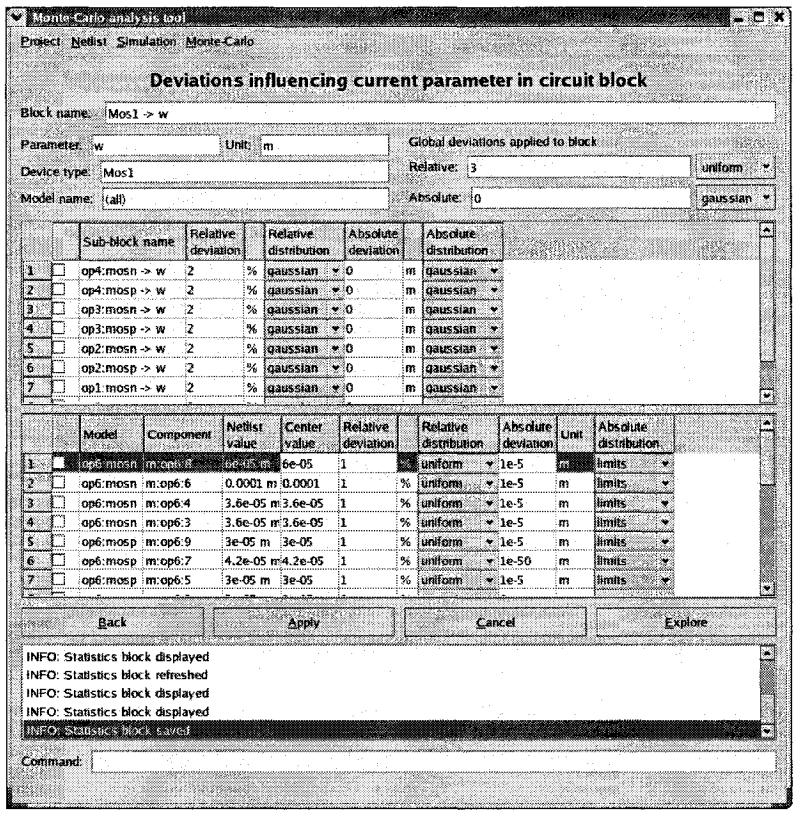


Figure 5.13 : GUI used to set parameter deviations in MC application

The specified number of circuit samples is generated according to the random deviations. For the simulation, the nominal circuit provides the AISA of each sample circuit. Once the simulations are performed on all circuit samples, the mean and standard deviation values are automatically computed for each simulation point. Moreover, acceptability thresholds can be defined to compute the percentage of acceptable circuit samples, which can be related to the robustness of the circuit and to manufacturing yield.

5.5 Experimental Results

This section presents the CPU time and accuracy performances achieved by *MultiSPICE* over *SPICE3f5* simulator in AFS, ACS and MC applications. Since *MultiSPICE* is based

on *SPICE3f5*, the CPU time gain over *SPICE3f5* is only due to the new techniques mentioned in this paper.

The experimental circuits used for simulation are from ITC'97 [27] and Circuitsim90 [28] sets of benchmark circuits. Those circuits are BJT or MOSFET transistor-level circuits. For each application, the following four DC simulation methods are compared:

1) Original *SPICE3f5* simulator.

2) *MultiSPICE* simulator without AISA: the gain for this method is only due to socket communications and circuit structure reutilization.

3) *MultiSPICE* simulator using AISA: an AISA is used and the iterations are conducted until convergence for accurate simulation results.

4) *MultiSPICE* with NR iterations ended before convergence using one of the methods proposed in section II.D: we used *TBSA* for AFS, *reltol* relaxation for ACS, and single iteration for MC applications.

For each method, *SPICE3f5* continuation methods are used if convergence is not reached after 100 iterations (*itl1* = 100). The CPU time is derived from the number of cycles the CPU spends working on the application, which excludes the lost time for file access. On the other hand, the real time is the time the user is effectively waiting for the results. To consider the files access delays, the gain achieved by *MultiSPICE* over *SPICE3f5* is computed using the real time. However, the different methods offered by *MultiSPICE* are compared on the CPU time basis, which takes advantage of higher time resolution. Hence, the gain of method (2) is computed as real time of (2) divided by real time of (1); the gain of method (3) as CPU time of (3) divided by CPU time of (2) multiplied by gain of (2), and so on. The simulations have been performed on a Pentium IV, 2 GHz, running Linux.

5.5.1 Analog Fault Simulation Results

Since circuit layout and process information are not available for the considered benchmark circuits, the list of faults is built from the netlist components. However, inductive fault analysis (IFA) is generally recommended to generate a realistic set of faults [29]. The list of parametric and catastrophic faults is summarized in Table 5.1. For open faults, the transistor is forced in cut-off region and resistive shorts R_o are added between transistor terminals.

Table 5.1 : Faults injected in the benchmark circuits

Device	Type	Value
Resistor	Short	R_s=1Ω, 500Ω
	Open	R_o=10MΩ
	Parametric (Resistance)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%
Capacitor	Short	R_s=1Ω, 500Ω
MOS	Short G-S, D-S, D-G	R_s=1Ω, 500Ω
	Open	Cut-off, R_o=10MΩ
	Parametric (W, L, tox)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%
BJT	Short B-E, C-E, C-B	R_s=1Ω, 500Ω
	Open	Cut-off, R_o=10MΩ
	Parametric (Area)	25%, 40%, 60%, 80%, 125%, 150%, 200%, 300%, 400%

Table 5.2 presents the DC fault simulation time of all faulty circuits generated for each benchmark circuit. A fault is considered detected if the output voltage of the faulty circuit differs from the mean value of fault-free output voltage by more than 3 times the standard deviation value of the fault-free output voltage. For each benchmark circuit, the mean and standard deviation of the fault-free output voltage are computed using MC analysis. In Table 5.2, the first column shows the circuit names whereas column 2 shows the number of faults to simulate. For each method, the required real or CPU time in seconds and their gain over *SPICE3f5* is presented. The "*Non conv.*" columns show the number of faulty

circuits which have not converged using *MultiSPICE* with and without AISA. Column "*Err*" shows the percentage of faults which are misclassified using *TBSA* method.

The results in Table 5.2 show that *MultiSPICE* reduces the DC simulation time by a factor of 58.6 over *SPICE3f5* simulator. The communication methodology and circuit reutilization offered by *MultiSPICE* achieves a gain of 11.6, whereas the use of an AISA gives an additional gain of 5.0 (58.6/11.6). The AISA also reduces the number of faulty circuits that cannot converge by a factor of 17 (88/1.8). The simulation robustness is therefore significantly improved. The use of *TBSA* method reduces the CPU time by an additional factor of 2.1, for a total average gain of 121 over *SPICE3f5*, with only 1.8 % of misclassified faults.

Table 5.2 : Simulation time speedup of *MultiSPICE* over *Spice3* in analog DC fault simulation

Circuit	Nb faults	Without AISA					MultiSPICE using AISA					
		SPICE3	MultiSPICE			Non	Convergence		TBSA			Non
		Real	Real	Cpu	Gain	conv.	Cpu	Gain	Cpu	Gain	%Err	conv.
leapfrog	1536	59	4	3.19	14.75	0	0.46	102.3	0.36	130.7	0.0	0
continous	810	32	7	5.75	4.57	102	0.68	38.7	0.32	82.1	1.9	3
ab_integ	883	35	2	1.79	17.50	0	0.65	48.2	0.29	108.0	1.0	0
ab_opamp	895	57	34	32.8	1.68	353	2.03	27.1	0.34	161.7	3.4	28
gm3	753	27	1	0.55	27.00	1	0.28	53.0	0.21	70.7	0.0	0
vreg	440	17	4	3.3	4.25	73	0.17	82.5	0.08	175.3	4.3	0
Average	886	38	8.7	7.90	11.62	88	0.71	58.6	0.27	121.4	1.8	5

5.5.2 Analog Circuit Sizing Results

The experimental results for ACS are presented in Table 5.3. The width of all MOSFETs and area factor of all BJT transistors are randomly modified by ±10 % of their nominal values to generate 1000 candidate circuits for each benchmark circuit. Parameter *reltol* is relaxed to 5 % for *reltol* relaxation method. The column 2 of Table 5.3 shows the number of parameters which are modified in each candidate circuit, which is equal to the number of transistors. Column "*Err*" shows the number of circuits which are not correctly

ordered using *reltol* relaxation method. If two circuits differ by less than accepted tolerance (2**reltol* = 10 %), they can be swapped in the ordering without considering an error. The other columns have the same meaning as in Table 5.3.

As presented in Table 5.3, *MultiSPICE* reduces the DC simulation time by an average factor of 106 over *SPICE3f5* using AISA. The *reltol* relaxation method gives an additional gain of 1.15, for a total average gain of 122 over *SPICE3f5*, whereas the ordering of the candidate circuits was not altered.

Table 5.3 : DC simulation time speedup of *MultiSPICE* over *Spice3* in analog circuit sizing for 1000 candidate circuits

Circuit	Nb. par.	Without AISA					MultiSPICE using AISA					
		SPICE3	MultiSPICE			Non conv.	Convergence		Reltol relaxation (5%)			Non conv.
		Real	Real	Cpu	Gain		Cpu	Gain	Cpu	Gain	%Err	
leapfrog	13	49	4	3.53	12.25	0	0.51	84.8	0.48	90.1	0.0	0
continous	27	46	12	11.2	3.83	175	0.45	95.4	0.42	102.2	0.0	0
ab_integ	31	45	2	1.63	22.50	0	0.57	64.3	0.4	91.7	0.0	0
ab_opamp	31	73	35	34.04	2.09	387	0.47	151.1	0.44	161.4	0.0	0
gm3	30	39	1	0.65	39.00	0	0.4	63.4	0.37	68.5	0.0	0
vreg	20	42	9	8.33	4.67	170	0.22	176.7	0.18	216.0	0.0	0
Average	25	49	10.5	9.90	14.06	122	0.44	105.9	0.38	121.6	0.0	0

5.5.3 Monte Carlo Analysis Results

Table 5.4 presents the MC simulation time on 1000 samples. The considered parameters for modifications are the following: MOS width and length, BJT area factor, resistance and capacitance values. A Gaussian distribution is used for each of the generated random values. Since no process information is available on the considered benchmark circuits, parameter deviations have been defined in such a way to obtain at least 95 % of the samples within ±10 % of the output value of the nominal circuit. Column 2 shows the number of parameters which are altered in each sample, whereas the "*Err*" column shows the error on the computed standard deviation value with the single iteration method. The other columns have the same meaning as in Table 5.2.

Table 5.4 shows that using *MultiSPICE* and AISA reduces the DC simulation time by an average factor of 90 over *SPICE3f5*. Ending DC simulation after a single iteration reduces the CPU time by an additional factor of 1.46 over full accuracy, for a total average gain of 131, without altering noticeably the computed mean and standard deviation.

Table 5.4 : DC simulation time speedup of *MultiSPICE* over *Spice3* in Monte Carlo analysis for 1000 circuit samples

Circuit	Nb. par.	Without AISA					MultiSPICE using AISA					
		SPICE3	MultiSPICE			Non conv.	Convergence		Single iteration			Non conv.
		Real	Real	Cpu	Gain		Cpu	Gain	Cpu	Gain	%Err	
leapfrog	121	57	3	2.06	19.00	0	0.66	59.3	0.58	67.5	0.0	0
continous	64	48	5	4.07	9.60	43	0.5	78.1	0.39	100.2	0.1	0
ab_integ	65	47	2	1.68	23.50	0	0.65	60.7	0.41	96.3	2.2	0
ab_opamp	66	76	35	33.75	2.17	424	0.67	109.4	0.37	198.1	2.8	0
gm3	60	44	1	0.84	44.00	0	0.41	90.1	0.29	127.4	6.8	0
vreg	30	44	9	8.37	4.89	172	0.29	141.1	0.21	194.9	5.3	0
Average	68	53	9.2	8.46	17.19	107	0.53	89.8	0.38	130.7	2.9	0

5.5.4 Summary

MultiSPICE simulator reduces significantly the DC simulation time required by the three studied applications. The use of socket communication instead of disk files reduces the simulation time by an average factor of 14. An additional average gain of 6 is obtained using AISA as the starting point of DC simulation. Hence, *MultiSPICE* is 85 times faster than *SPICE3f5* while producing the same simulation results. In addition, the number of non convergent circuits is globally reduced from 10 to 0.2 %, which shows a major improvement on the simulation robustness. Also, the methods offered by *MultiSPICE* to end simulation before convergence allows an additional simulation time gain. Indeed, AFS using *TBSA* method, ACS using *reltol* relaxation method and MC using single iteration method are respectively 121, 122 and 131 times faster than *SPICE3f5* simulator. Compared to accurate simulation, only an average of 1.8 % of the faulty circuits are misclassified for AFS application, the candidate circuits' ordering remains unchanged for

ACS application, and for MC analysis, the obtained standard deviation results differ by less than 3 % in average. These differences remain negligible compared to the obtained CPU time gain.

5.6 Conclusion

In this paper, we proposed new simulation techniques to reduce the CPU time required by the simulation of a large number of similar circuits. *MultiSPICE* is a SPICE-like simulator optimized for the simulation of multiple circuits. *MultiSPICE* takes advantage of 4 concepts: 1) socket connection with the calling application, 2) efficient injection of modifications in nominal circuit, 3) use of accurate initial DC solution approximations and 4) methods to end DC simulation before convergence

MultiSPICE simulator has been successfully exploited in analog fault simulation, circuit sizing and Monte Carlo applications. Experimental results show that *MultiSPICE* is 85 times faster than *SPICE3f5* in DC simulation without impacting the results accuracy. The use of proposed methods to end NR iterations before convergence reduced the simulation time by a factor of 125 compared to *SPICE3f5*, without altering the behavior of the considered applications.

Although most improvements are dedicated to DC simulation, we observed that in transient simulation, *MultiSPICE* is 2 to 3 times faster than *SPICE3f5*. Therefore, in our future works we will work on new methods that could reduce the CPU time involved in transient simulation. As we did in DC simulation, we aim to exploit efficiently the similarities of the results between each of the modified circuits. Moreover, *MultiSPICE* could also be improved to support the distribution of the simulations on two or more computers communicating over a network. The new socket communications through TCP protocol proposed in this paper would facilitate the implementation of such a distribution.

5.7 References

[1] G. Van der Plas, G. Debyser, F. Leyn, K. Lampaert, J. Vandenbusshe, G.G.E. Gielen, W. Sansen, P. Veselinovic, D. Leenaerts, "AMGIE – A synthesis environment for CMOS analog integrated circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1037-1058, Sept. 2001.

[2] *STAR-HSPICE Quick Reference Guide 2001.4*, Avanti Co., Fremont, California, U.S.A., Dec. 2001.

[3] *SPECTRE Circuit Simulator User Guide, Product version 5.0*, Cadence Design Systems inc., San Jose, California, U.S.A., June 2003.

[4] *Eldo User's Manual, version 6.5_2*, Mentor Graphics Corp., Wilsonville, Oregon, U.S.A., 2005.

[5] S. Spinks, I. Bell, "ANTICS analogue fault simulation software," *IEE Colloquium on Testing Mixed Signal Circuits and Systems*, pp. 13/1-13/5, Oct. 1997.

[6] Z.R. Yang, M. Zwolinski, "Fast, robust DC and transient fault simulation for nonlinear analogue circuits," *Proc. Design, Automation and Test in Europe Conf. and Exhibition*, pp. 244-248, March 1999.

[7] M.W. Tian, C.-J.R. Shi, "Efficient DC fault simulation of nonlinear analog circuits," *Proc. Design, Automation and Test in Europe*, pp. 899-904, Feb. 1998.

[8] J. Hou, A. Chatterjee, "Concurrent transient fault simulation for analog circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1385-1398, Oct. 2003.

[9] M.W. Tian, C.-J.R. Shi, "Nonlinear analog DC fault simulation by one-step relaxation," *Proc. 16th IEEE VLSI Test Symposium*, pp. 126-131, April 1998.

[10] P.N. Variyam, J. Hou, A. Chatterjee, "Test generation for analog circuits using partial numerical simulation," *Proc. 12th Int. Conf. on VLSI Design*, pp. 597-602, Jan. 1999.

[11] G.G.E. Gielen, R.A. Ruterbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. of the IEEE*, vol. 88, no. 12, pp. 1825-1854, Dec. 2000.

[12] R. Phelps, M. Krasnicki, R.A. Rutenbar, L.R. Carley, J.R. Hellums, "Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 6, pp. 703-717, June 2000.

[13] G. Alpaydin, S. Balkir, G. Dundar, "An evolutionary approach to automatic synthesis of high-performance analog integrated circuits," *IEEE Trans. Evolutionary Computation*, vol. 7, no. 3, pp. 240-252, June 2003.

[14] F. Meideiro, F.V. Fernandez, R. Dominguez-Castro, A. Rodrigez-Vazkez, "A Statistical optimization-based approach for automated sizing of analog cells," *Proc. Int. Conf. Computer-Aided Design*, pp. 594-597, Nov. 1994.

[15] R. Iskander, M. Dessouky, M. Aly, M. Magdy, N. Hassan, N. Soliman, S. Moussa, "Synthesis of CMOS analog cells using AMIGO," *Proc. Design, Automation and Test in Europe*, pp. 297-302, 2003.

[16] H. Su, C. Michael, M. Ismail, "Yield optimization of analog MOS integrated circuits including transistor mismatch," *IEEE Int. Symposium on Circuits and Systems*, vol. 3, pp. 1804-1804, May 1993.

[17] R.O. Topaloglu, "Monte Carlo-Alternative Probabilistic Simulations for Analog Systems," *Proc. 7th Int. Symposium on Quality Electronic Design*, March 2006.

[18] S.J. Spinks, C.D. Chalk, I.M. Bell, M. Zwolinski, "Generation and verification of tests for analog circuits subject to process parameter deviations," *Journal of Electronic Testing: Theory and Applications*, vol. 20, no. 1, pp. 11-23, 2004.

[19] A. Khouas, A. Derieux, "Fault simulation for analog circuits under parameter variations," *Journal of Electronic Testing: Theory and Applications*, vol. 16, no. 3, pp. 269-278, 2000.

[20] Z. Wang, S.W. Director, "An efficient yield optimization method using a two step linear approximation of circuit performance," *Proc. European Design and Test Conf.*, pp. 567-571, March 1994.

[21] R. Lopez-Ahumada, R. Rodriguez-Macias, "FASTEST: A tool for a complete and efficient statistical evaluation of analog circuits. DC analysis," *Analog Integrated Circuits and Signal Processing*, vol. 29, no. 3, pp. 201-212, Dec. 2001.

[22] J. Vlach, K. Singhal, *Computer Methods for Circuit Analysis and Design*, Van Nostrand Reinhold Company, U.S.A., 1983.

[23] T. Quarles, A.R. Newton, D.O. Pederson, A.S. Vincentelli, *SPICE3F User's Manual*, University of California at Berkeley, California, U.S.A., May 1993.

[24] M. Morneau, A. Khouas, "Analysis of DC simulation convergence of nonlinear analog circuits with initial solution," *IEEE Canadian Conf. on Electrical and Computer Engineering*, pp. 708-712, May 2005.

[25] R.C. Melville, L.Trajkovic, S.-C. Fang, L.T. Watson, "Artificial parameter homotopy methods for the DC operating point problem," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, June 1993.

[26] M. Morneau, A. Khouas, "TBSA: Threshold-based simulation accuracy method for fast analog DC fault simulation," to appear in *Journal of Electronic Testing: Theory and Applications*.

[27] B. Kaminska, K. Arabi, I. Bell, P. Goteti, J. L. Huertas, B. Kim, A. Rueda, M. Soma, "Analog and mixed-signal benchmark circuits – first release," *Proc. Int. Test Conf.*, pp. 183-190, Nov. 1997.

[28] "Circuitsim90," *1990 Circuit Simulation and Modeling Workshop at MCNC.*

[29] M. Sachdev, B. Atzema, "Industrial relevance of analog IFA: a fact or a fiction," *Proc. Int. Test Conf.*, pp. 61-70, Oct. 1995.

CHAPITRE 6

Discussion générale

Ce chapitre reprend les éléments de discussion les plus importants pour les méthodes d'accélération de la simulation DC analogique présentées dans ce mémoire. L'importance de la simulation DC pour les applications visées est également soulignée.

Tout d'abord, le chapitre 2 présentait une analyse de la convergence de l'algorithme NR dans le cas de la simulation DC. L'importance de l'utilisation d'une AIPS a été mise en évidence. La méthode d'augmentation de la tolérance relative a été proposée comme méthode d'arrêt des itérations avant la convergence. Les itérations sont terminées lorsque la précision requise est atteinte. Cette méthode permet un bon compromis entre le temps de calcul et la précision des résultats de simulation.

Les résultats expérimentaux obtenus démontrent que cette méthode fonctionne très bien lorsque le circuit est soumis à de légères variations à ses paramètres. L'AIPS utilisé se rapproche dans ce cas de la solution finale. De façon générale, on peut dire que la méthode d'augmentation de la tolérance est pertinente lorsqu'un a une bonne idée de la solution finale mais qu'on doit obtenir une certaine précision. Le nombre d'itérations NR requises peut être réduit de 20 à 40 % avec un faible risque d'erreurs. Cette méthode est toutefois déconseillée dans le cas où la solution finale est très difficilement prévisible puisque les itérations peuvent être terminées prématurément, ce qui génère une solution erronée. De plus, tant pour une faible (20 %) qu'une moyenne (5 %) précision requise, le risque d'erreurs demeure inférieur à 1 %. Le nombre d'itérations est bien entendu plus faible pour une moyenne précision, mais le risque d'obtenir des résultats ne respectant pas la précision requise s'en trouve réduit.

Pour la méthode d'augmentation de la tolérance relative, la précision de l'approximation initiale de la solution ainsi que la valeur de précision désirée (*reltol*) sont les paramètres qui influencent le gain en terme d'itérations requises et le risque d'obtention de résultats erronés. Cette méthode peut être utilisée dans n'importe quelle application nécessitant des simulations DC multiples et pour laquelle les résultats de simulation peuvent être précis à un certain pourcentage près.

L'article présenté au chapitre 4 proposait une nouvelle méthode d'arrêt des itérations NR avant la convergence. Celle-ci, appelée *TBSA*, a été appliquée à la simulation DC de pannes analogiques. Comme pour la précédente, cette méthode nécessite tout d'abord une AIPS à partir de laquelle l'algorithme NR est démarré. Les itérations sont ensuite terminées dès que la tension de sortie est suffisamment précise pour que la panne puisse être classifiée comme détectée ou non. La métrique *VarRelMax* est utilisée à chaque itération afin d'approximer la précision de la solution intermédiaire obtenue. Tant pour les pannes paramétriques que catastrophiques, un compromis entre le nombre d'itérations NR et la précision des résultats est réalisé.

Nos résultats expérimentaux démontrent que la méthode *TBSA* est beaucoup plus rapide que la simulation jusqu'à la convergence tout en présentant un risque d'erreur beaucoup plus faible que l'arrêt de la simulation après une seule itération. Si une bonne AIPS est utilisée, par exemple à partir d'une interpolation linéaire, il est possible d'éliminer presque complètement le risque de mauvaise classification des pannes paramétriques. D'ailleurs, l'amélioration de l'AIPS réduit le nombre d'itérations NR requises ainsi que le risque de mauvaise classification de pannes. Il est donc possible de réduire le nombre d'itérations d'un facteur de 2.65 par rapport à la simulation jusqu'à la convergence sans risque important de mauvaise classification de pannes. Toutefois, dans le cas des pannes catastrophiques, il est plus difficile de prévoir la solution finale puisque la structure du circuit est altérée. Par conséquent, le risque de mauvaise classification de pannes de la méthode *TBSA* est plus élevé que dans le cas des pannes paramétriques. Par contre, ce

risque est trois fois plus faible qu'avec la méthode qui consiste à effectuer une seule itération, ce qui démontre la supériorité de la méthode *TBSA* sur cette dernière.

Une méthodologie de test nécessite généralement la simulation de chaque panne sous plusieurs stimuli afin de connaître quels stimuli peuvent détecter chaque panne. Dans ce cas, le risque qu'une mauvaise classification unique se retrouve dans la suite finale de tests s'en trouve réduit. En d'autres mots, même si une panne est mal classifiée pour un stimulus, elle a de fortes chances d'être correctement classifiée pour le stimulus suivant, ce qui permet de rejeter la mauvaise classification initiale. Ainsi, le risque d'erreur introduit par *TBSA* s'en trouve réduit. La réduction du temps de calcul apportée par la méthode *TBSA* peut permettre de simuler une liste plus exhaustive de pannes ou de simuler les pannes sous un plus grand nombre de stimuli dans un temps donné, ce qui améliore globalement la méthodologie de test.

La méthode *TBSA* ne peut être utilisée dans un simulateur standard puisque les seuils de détection de pannes sont requis et les solutions intermédiaires à chaque itération doivent être disponibles afin de calculer *VarRelMax* et ainsi terminer les itérations lorsque la panne peut être classifiée. De plus, la nécessité de démarrer la simulation à partir d'une AIPS est difficile à supporter avec un tel simulateur. Par conséquent, un simulateur doit être adapté pour supporter cette méthode, ce qui est le cas avec *MultiSPICE*.

L'article du chapitre 5 présentait un simulateur analogique optimisé pour les applications nécessitant des simulations multiples d'un circuit avec de légères modifications. Le simulateur *MultiSPICE* permet en effet d'accélérer de façon importante les applications de simulation de pannes, de dimensionnement automatique de circuits et d'analyse Monte Carlo sans altérer leur fonctionnement. Plutôt que d'être invoqué à chaque simulation, le simulateur demeure actif tant et aussi longtemps que d'autres simulations peuvent être requises. Une connexion *socket* permet à l'application de spécifier des circuits modifiés au simulateur et de récupérer ensuite les résultats de simulation. Les simulations DC sont

démarrées à partir d'une AIPS judicieusement calculée à partir des résultats des simulations précédentes, selon une équation fournie par l'application. Enfin, *MultiSPICE* propose les méthodes suivantes permettant d'arrêter les itérations avant la convergence : nombre fixe d'itérations [65], augmentation de la tolérance relative (chapitre 2) et *TBSA* (chapitre 4), cette dernière étant étendue aux applications qui doivent comparer des circuits entre eux. Des informations détaillées sur le fonctionnement du simulateur *MultiSPICE* sont présentées à l'annexe B.

Des applications de simulation de pannes, de dimensionnement de circuits et d'analyse Monte Carlo ont été développées afin de valider les méthodes d'accélération proposées. Ces applications sont décrites plus en détails à l'annexe C. Un résumé des gains en temps de calcul obtenus est présenté à la Figure 6.1 (a). Les résultats sont assez similaires peu importe l'application considérée, ce qui démontre que *MultiSPICE* procure des gains significatifs dans les trois applications visées. Globalement, l'utilisation d'un lien de communication *socket* entre l'application et le simulateur ainsi que l'injection rapide des modifications dans le circuit nominal réduisent le temps d'attente des résultats de simulation DC d'un facteur de 14 d'après la Figure 6.1 (b). Notre hypothèse selon laquelle les accès au disque dur et la reconstruction du circuit en mémoire constituent un goulot d'étranglement dans les applications visées est donc vérifiée.

Les résultats expérimentaux obtenus montrent également que le temps de calcul peut être réduit d'un facteur de 6 lorsqu'une AIPS est utilisée selon la Figure 6.1 (b). *MultiSPICE* a l'avantage important de permettre à l'application de spécifier l'équation qui permet de calculer cet AIPS. Ainsi, plus l'application est en mesure de fournir une bonne approximation des solutions attendues, meilleur est le gain en temps de calcul obtenu et ce, de façon transparente par rapport à *MultiSPICE*. De plus, l'utilisation d'une AIPS permet de réduire significativement le risque de non convergence d'un circuit, c'est-à-dire le risque qu'aucun résultat ne soit retourné par le simulateur. Nos résultats expérimentaux montrent qu'environ 11 % des circuits modifiés ne peuvent converger

sans AIPS, alors que cette proportion descend à 0.2 % avec une AIPS. Le risque de non convergence est donc réduit d'un facteur de 50, augmentant par le fait même la robustesse du simulateur. En additionnant les gains dus aux communications *socket* et à l'utilisation d'une AIPS, *MultiSPICE* est 85 fois plus rapide que *SPICE3f5* sans que les résultats de simulation ne soient affectés. En d'autres mots, sans aucun compromis sur la précision.

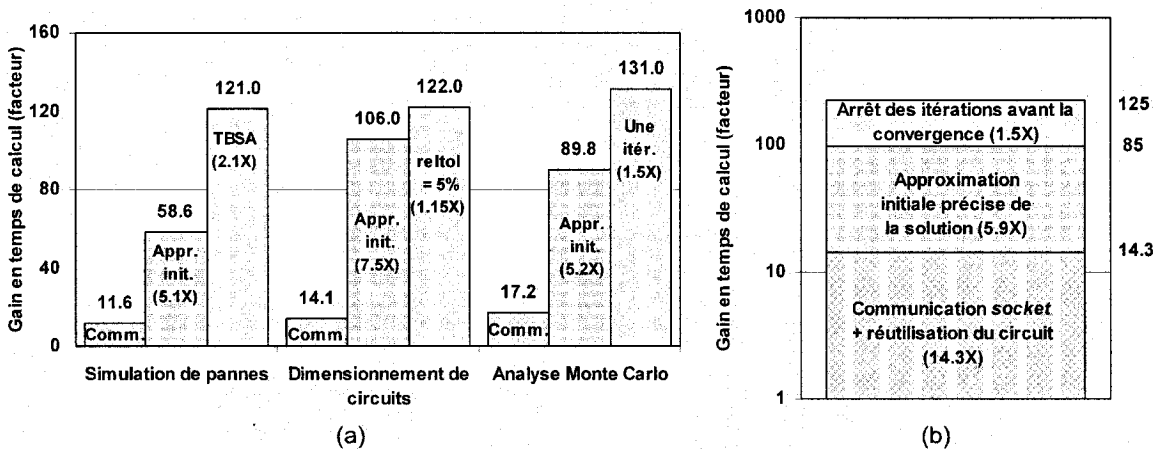


Figure 6.1 : Gain en temps de calcul de MultiSPICE par rapport à SPICE3f5 (a) pour chaque application et (b) cumulatif, applications confondues

Si l'application permet un compromis entre le temps de calcul et la précision des résultats, le temps de calcul peut être réduit d'un facteur additionnel de 1.5 avec un très faible risque de résultats de simulation erronés. Parmi les trois méthodes d'arrêt des itérations avant la convergence expérimentées, la méthode *TBSA* en simulation de pannes est celle qui produit le gain le plus significatif avec 2.1. Ce gain en temps de calcul est un peu inférieur à celui en nombre d'itérations (environ 3) présenté au chapitre 4, qui est le cas idéal. Cette différence s'explique par le fait que la quantité de données transmises à *MultiSPICE* et de résultats reçus par la suite est identique peu importe la méthode d'arrêt des itérations utilisée. Il en est de même pour les opérations effectuées lors de l'injection des modifications dans le circuit nominal ainsi qu'à la récupération des résultats. Par conséquent, pour une réduction d'un facteur de 3 du nombre d'itérations, la réduction du

temps d'attente des résultats est plutôt d'un facteur de l'ordre de 2.1. Le même raisonnement s'applique pour la méthode d'augmentation de la tolérance relative en dimensionnement de circuits, qui réduit le nombre d'itérations de l'ordre de 30 % mais réduit le temps de calcul de 15 %. Enfin, comme le nombre d'itérations NR requises pour atteindre la convergence est généralement faible en analyse Monte Carlo, la méthode d'une seule itération réduit le temps de calcul d'environ 50 %, c'est-à-dire moins que la méthode *TBSA*.

Les méthodes proposées dans le cadre de ce mémoire visaient de façon particulière la simulation DC. Il est évident que l'analyse DC n'est pas suffisante à elle seule pour détecter tous les défauts de fabrication possibles, mesurer toutes les performances à atteindre par un circuit ainsi que pour mesurer tous les effets des fluctuations du procédé de fabrication. Toutefois, les pannes qui sont détectées en analyse DC n'ont plus besoin d'être simulées en analyses AC et transitoire, beaucoup plus coûteuses en temps de calcul [58]. De même, si le point de polarisation d'un circuit candidat pour le dimensionnement est éloigné du point attendu, il est inutile de rechercher des caractéristiques transitoires complexes comme le temps de réponse et la marge de phase. Dans ces cas, la réduction du temps de calcul en analyse DC a un impact non négligeable sur l'application considérée.

Le point de polarisation DC est également utilisé comme point de départ des analyses AC et transitoire. La diminution du temps de calcul nécessité par l'analyse DC entraîne donc une diminution dans une plus faible proportion du temps requis par les analyses AC et transitoires. Des résultats préliminaires montrent que *MultiSPICE* réduit le temps de calcul en analyse transitoire d'un facteur de l'ordre de 2 à 3. Bien entendu, si un circuit est simulé durant un très long intervalle de temps, ce gain devient de plus en plus marginal. C'est pourquoi de nouvelles méthodes visant spécifiquement l'analyse transitoire doivent être développées.

Conclusion et recommandations

Avec la complexité croissante des circuits analogiques et mixtes, le besoin d'outils CAO performants devient de plus en plus important. De nombreux travaux de recherche se sont intéressés au développement de méthodes de simulation de circuits analogiques précises et fiables. Toutefois, de nombreuses tâches reliées à la conception de circuits analogiques ne sont toujours pas automatisées. Ce projet de recherche visait à diminuer le temps de calcul de certaines applications qui font appel de façon répétitive à un simulateur analogique. Trois applications ont été ciblées par ce mémoire, à savoir la simulation de pannes, le dimensionnement automatique de circuits et l'analyse Monte Carlo.

Une analyse du comportement de l'algorithme NR à l'aide d'exemples expérimentaux nous a permis de mieux comprendre la dynamique de convergence d'un circuit analogique. La métrique de tolérance relative maximale, qui permet d'approximer la précision atteinte à chaque itération, a ensuite été proposée. L'augmentation du paramètre de tolérance relative (*reltol*) des simulateurs SPICE jusqu'à la précision requise par l'application permet de terminer la simulation DC dès que cette précision est atteinte. Cette méthode, qui nécessite une AIPS, a réduit le nombre d'itérations de 20 à 40 % sur des circuits de référence, dépendamment des modifications injectées et de la précision requise.

Nous avons également proposé une méthode d'arrêt des itérations avant la convergence qui s'applique bien à la simulation DC de pannes analogiques. La méthode *TBSA* permet de terminer les itérations dès que la précision est suffisante pour classifier la panne comme détectée ou non. La méthode *TBSA* a permis de réduire d'un facteur de 3 environ le nombre d'itérations NR par rapport à la simulation jusqu'à la convergence sur des circuits de référence. Toute méthode susceptible d'améliorer la précision de l'AIPS

utilisée améliore également les performances de *TBSA*, tant en ce qui concerne la réduction du nombre d'itérations que le risque d'erreurs de classification de pannes. De plus, cette méthode peut être étendue à toute application qui nécessite la comparaison de circuits, comme le dimensionnement de circuits.

Le simulateur analogique *SPICE3f5* a également été optimisé pour les applications nécessitant des simulations multiples. Quatre améliorations sont proposées : une connexion *socket* avec l'application appelante, l'injection efficace des modifications dans le circuit nominal, l'utilisation d'AIPS et l'utilisation de méthodes d'arrêt de la simulation DC avant la convergence. Des applications de simulation de pannes, de dimensionnement de circuits et d'analyse Monte Carlo offrant une interface graphique utilisateur ont été développées afin d'évaluer les performances du simulateur proposé, *MultiSPICE*. Les résultats expérimentaux obtenus sur un ensemble de circuits de référence démontrent que *MultiSPICE* est en moyenne 85 fois plus rapide que *SPICE3f5* pour effectuer les simulations multiples en analyse DC, tout en produisant des résultats identiques. Si une méthode d'arrêt des itérations avant la convergence est utilisée, *MultiSPICE* est alors 125 fois plus rapide, alors que la proportion de résultats erronés est très faible. De plus, le nombre de circuits qui ne convergeaient pas dans *SPICE3f5* a été réduit d'un facteur de 50 grâce à l'utilisation d'AIPS. En ce qui concerne l'analyse transitoire, qui ne tire pas partie des méthodes d'arrêt des itérations avant la convergence, *MultiSPICE* permet tout de même de réduire le temps de calcul d'un facteur de 2 à 3.

L'objectif d'une réduction significative du temps de calcul nécessité par les simulations multiples a été atteint dans le cadre de ce mémoire. Deux méthodes d'arrêt des itérations avant la convergence en simulation DC ont été proposées et un simulateur optimisé pour les simulations multiples a été développé. Il serait sans doute possible d'inclure l'ensemble des méthodes suggérées à l'intérieur d'un simulateur commercial comme *Eldo*, *HSPICE* et *Spectre*. Souhaitons que certaines des méthodes proposées dans ce

mémoire inspireront un jour les concepteurs d'outils de CAO afin d'accélérer les simulateurs commerciaux pour les applications nécessitant des simulations multiples.

Quelques avenues seraient intéressantes à explorer suite aux travaux effectués dans le cadre de ce mémoire. Des résultats intermédiaires comme le système matriciel MNA pourraient être conservés à chaque itération et d'un circuit à l'autre. Il serait alors possible de réduire grandement le nombre de composants non linéaires à évaluer s'il est prévu que leurs résultats demeureront identiques à ceux de l'itération précédente ou du circuit précédent [22] et [68]. Une technique intermédiaire pourrait consister à remplacer de façon automatique les équations complexes des transistors par des fonctions simples, comme des fonctions linéaires par partie, dans le cas où on s'attend à ce que les résultats changent peu. Le temps d'évaluation des composants du circuit serait alors grandement réduit. De plus, diverses techniques ont été proposées afin d'accélérer la factorisation LU dans le cas où plusieurs résultats dans le système linéaire n'ont pas été modifiés par rapport à l'itération précédente [22], [12], [9] et [27]. Également, l'algorithme de *Crout* qui résout le système linéaire par factorisation LU peut être avantageusement remplacé par l'algorithme itératif *Gauss-Seidel* dans le cas où on s'attend à une solution semblable à l'itération NR précédente [5]. Le temps de résolution du système d'équations linéaires obtenu à chaque itération NR peut alors être significativement réduit.

Il serait certainement possible d'exploiter certaines de ces méthodes, qui ont surtout été étudiées en simulation de pannes, dans le cas des simulations multiples. Elles ont l'avantage d'accélérer de façon significative chacune des itérations NR, ce qui n'a pas été abordé dans ce projet. D'ailleurs, ces méthodes s'appliquent très bien à l'analyse transitoire, qui permet un partage de l'information entre les points temporels. Un compromis entre le temps de calcul et la précision des résultats pourrait alors être obtenu. Il serait donc envisageable de développer une méthode plus rapide qu'une seule itération NR. Une forme d'arrêt des itérations avant la convergence en simulation transitoire pourrait aussi être envisagée.

Dans un autre ordre d'idées, la simulation des circuits modifiés pourrait être distribuée efficacement sur un réseau d'ordinateurs afin de réduire le temps d'attente des résultats. En effet, l'architecture de communication proposée dans ce mémoire, qui fait appel à une connexion *socket*, est appropriée pour la communication entre plusieurs machines. Les résultats de simulation pourraient alors être automatiquement communiqués d'une machine à l'autre lorsque ces résultats sont requis pour le calcul d'une AIPS, par exemple. Un module permettant de distribuer les circuits à simuler et de gérer l'information à partager entre les machines devrait alors être ajouté aux applications développées, alors que certaines fonctions supplémentaires devraient être supportées par *MultiSPICE*.

Les applications de simulation de pannes, de dimensionnement de circuits et d'analyse Monte Carlo ont été conçues de façon à être indépendantes du simulateur utilisé. Le développement d'interfaces de communication entre ces applications et les simulateurs commerciaux *HSPICE* et *Spectre* pourrait permettre de continuer à utiliser et à développer les applications. Parmi les améliorations possibles à apporter aux applications développées, notons la génération automatique de suites de tests pour la simulation de pannes d'un circuit analogique donné. Un algorithme complet de dimensionnement de circuits ou de synthèse analogique permettant de trouver automatiquement les dimensions optimales d'un circuit donné pourrait être développé. Une application permettant l'analyse des disparités entre les composants du circuit (*mismatch*) pourrait également être développée à partir de l'outil d'analyse Monte Carlo. Ces applications pourraient aussi être améliorées de façon à être en mesure de fournir une AIPS plus précise au simulateur *MultiSPICE* et ainsi améliorer ses performances.

Bibliographie

[1] ACCELLERA INTERNATIONAL INC, *Verilog AMS language reference manual*, version 2.2, 2004. [En ligne] http://www.eda.org/verilog-ams (Consulté le 27 octobre 2006).

[2] ALPAYDIN, G., BALKIR, S., DUNDAR, G., "An evolutionary approach to automatic synthesis of high-performance analog integrated circuits," *IEEE Trans. Evolutionary Computation*, vol. 7, no. 3, pp. 240-252, juin 2003.

[3] AUGUSTO, J.S., ALMEIDA, C.F.B., "Circuit equations for fast fault simulation and diagnosis of linear circuits," in *Proc. IEEE International Conference on Electronics, Circuits and Systems*, 1998, vol. 1, pp. 125-128.

[4] AVENTI CO., *Star-Hspice Quick Reference Guide, Release 2001.4*, Fremont, California, 2001.

[5] BURCH, R., YANG, P., COX, P., MAYARAM, K., "A new matrix solution technique for general circuit simulation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 225-241, fév. 1993.

[6] CADENCE DESIGN SYSTEMS INC, *Spectre Circuit Simulator User Guide, Product version 5.0*, San Jose, California, 2003.

[7] CROW, M., *Computational methods for electric power systems*, CRC Press LLC, États-Unis, 2003.

[8] DABROWSKI, J., "Functional-level analogue macromodelling with piecewise linear signals," in *Proc. IEE Proceedings of Circuits, Devices-Level and Systems*, 1999, vol. 146, no. 2, pp. 77-82.

[9] DAVIS, A., "Acceleration of analog simulation by partial LU decomposition," in *Proc. 39th IEEE Midwest Symposium on Circuits and Systems*, 1996, vol. 1, pp. 335-338.

[10] DEGRAUWE, M., NYS, O., DIJKSTRA, E., RIJMENANTS, J., BITZ, S., GOFFART, B.L.A.G., VITTOZ, E.A., CSERVENY, S., MEIXENBERGER, C., STAPPEN, G.V.D., OGUEY, H.J., "IDAC: An interactive design tool for analog CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 1106-1116, déc. 1987.

[11] DEVARAYANADURG, G., GOTETI, P., SOMA, M., "Hierarchy based statistical fault simulation of mixed-signal ICs," in *Proc. International Test Conference*, 1996, pp. 521-527.

[12] ENGIN, N., KERKHOFF, H.G., "Fast fault simulation for nonlinear analog circuits," *IEEE Design & Test of Computers*, vol. 20, no. 2, pp. 40-47, mars/avril 2003.

[13] FANG, L., GRONTHOUD, G., KERKHOFF, H.G., "Reducing analogue fault simulation time by using high-level modeling in Dotss for an industrial design," in *Proc. IEEE European Test Workshop*, 2001, pp. 61-67.

[14] FORTIN, A., *Analyse numérique pour ingénieurs*, Éditions de l'École Polytechnique de Montréal, Canada, 1996.

[15] GIELEN, G.G.E., RUTERBAR, R.A., "Computer-aided design of analog and mixed-signal integrated circuits," in *Proc. of the IEEE*, 2000, vol. 88, no. 12, pp. 1825-1854.

[16] HAMIDA, N.B., SAAB, K., MARCHE, D., KAMINSKA, B., QUESNEL, G., "LIMSoft: Automated tool for design and test integration of analog circuits," in *Proc. International Test Conference*, 1996, pp. 571-580.

[17] HAN, D., CHATTERJEE, A., "Simulation-in-the-loop analog circuit sizing method using adaptative model-based simulated annealing," in *Proc. 4th IEEE International Workshop on Systems on Chip for Real-Time Application*, 2004, pp. 127-130.

[18] HARJANI, R., "OASYS: A framework for analog circuit systhesis," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 18, pp. 1247-1266, déc. 1989.

[19] HARVEY, J.P., ELMASRY, M.I., LEUNG, B., "STAIC: An interactive framework for synthesizing CMOS and BiCMOS analog circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 11, pp. 1402-1417, nov. 1992.

[20] HARVEY, R.J.A., RICHARDSON, A.M.D., BRULS, E.M.J.G., BAKER, K., "Analogue fault simulation based on layout dependant fault models," in *Proc. International Test Conference*, 1994, pp. 641-649.

[21] HOCHWALD, W., BASTIAN, J.D., "A DC approach for analog fault dictionary determination," *IEEE Trans. Circuits and Systems*, vol. 26, pp. 523-529, juil. 1979.

[22] HOU, J., CHATTERJEE, A., "Concurrent transient fault simulation for analog circuits », *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1385-1398, oct. 2003.

[23] HUYNH, S.D., KIM, S., SOMA, M., ZHANG, J., "Automatic analog test signal generation using multifrequency analysis," *IEEE Trans. Circuits and Systems: II – Analog and Digital Signal Processing*, vol. 46, no. 5, pp. 565-576, mai 1999.

[24] THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS INC, "IEEE Standard VHDL analog and mixed-signal extensions", *IEEE Std, 1076.1-1999*, 1999. [En ligne] http://ieeexplore.ieee.org/xpl/standards.jsp (Consulté le 27 octobre 2006).

[25] ISKANDER, R., DESSOUKY, M., ALY, M., MAGDY, M., HASSAN, N., SOLIMAN, N., MOUSSA, S., "Synthesis of CMOS analog cells using AMIGO," in *Proc. Design, Automation and Test in Europe*, 2003, pp. 297-302.

[26] KAMINSKA, B., ARABI, K., BELL, I., GOTETI, P., HUERTAS, J.L., KIM, B., RUEDA, A., SOMA, M., "Analog and mixed-signal benchmark circuits – first release," in *Proc. International Test Conference*, 1997, pp. 183-190. [Circuits en ligne] http://www.cs.wright.edu/~emmert/analog_bm_ckts/main_bm.htm (Consulté le 27 octobre 2006).

[27] KHER, S.S., CARTER, H.W., "Improving Analog Simulation Speed using Selective Matrix Update," in *Proc. International Workshop on Behavioral Modeling and Simulation*, 2003, pp. 97-101.

[28] KHOUAS, A., DERIEUX, A., "Fault simulation for analog circuits under parameter variations," *Journal of Electronic Testing: Theory and Applications*, vol. 16, no. 3, pp. 269-278, juin 2000.

[29] KIELKOWSKI, R.M., *Inside Spice Overcoming the obstacles of circuit simulation*, McGraw-Hill, Inc, U.S.A., 1994.

[30] KILIC, Y., ZWOLINSKI, M., "Behavioral fault modeling and simulation using VHDL-AMS to speed-up analog fault simulation," *Analog Integrated Circuits and Signal Processing*, vol. 39, no. 2, mai 2004.

[31] KOH, H.Y., SÉQUIN, C.H., "OPASYN: A compiler for CMOS operational amplifiers," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 2, pp. 113-125, fév. 1990.

[32] KOZA, J.R., BENNETT, F.H., ANDRE, D., KEANE, M.A., DUNLAP, F., "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Trans. Evolution Computations*, vol. 1, no. 2, pp. 109-128, juil. 1997.

[33] KRASNICKI, M., PHELPS, R., RUTENBAR, R.A., CARLEY, L.R., "MAELSTROM: Efficient simulation-based synthesis for custom analog cells," in *Proc. 36th Design Automation Conference*, 1999, pp. 945-950.

[34] LEENAERTS, D.M.W., VAN SPAANDONK, J., "DC testing of analog integrated circuits with piecewise linear approximation and interval analysis," in *Proc. IEEE International Symposium on Circuits and Systems*, 1993, pp. 1337-1340.

[35] LITOVSKI, V.B., LITOVSKI, I.V., ZWOLINSKI, M., "Concurrent analogue fault simulation, the equation formulation aspect," *International Journal of Circuit Theory and Applications*, vol. 32, no. 6, pp. 487-507, nov./déc. 2004.

[36] LOPEZ-AHUMADA, R., RODRIGEZ-MACIAS, R., "FASTEST: A tool for a complete and efficient statistical evaluation of analog circuits. DC analysis," *Analog Integrated Circuits and Signal Processing*, vol. 29, no. 3, pp. 201-212, déc. 2001.

[37] MCNC 1990. "Circuitsim90", *Circuit Simulation and Modeling Workshop at MCNC*. [En ligne] http://www.cbl.ncsu.edu:16080/benchmarks/CircuitSim90/ (Consulté le 27 octobre 2006).

[38] MEIDEIRO, F., FERNANDEZ, F.V., DOMINGUEZ-CASTRO, R., RODRIGEZ-VAZKEZ, A., "A statistical optimization-based approach for automated sizing of analog cells," in *Proc. IEEE International Conference on Computer-Aided Design*, 1994, pp. 594-597.

[39] MELVILLE, R.C., TRAJKOVIC, L., FANG, S.-C., WATSON, L.T., "Artificial parameter homotopy methods for the DC operating point problem," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, juin 1993.

[40] MENTOR GRAPHICS CORP. *Eldo User's Manual, version 6.5_2*, Wilsonville, Oregon, États-Unis, 2005.

[41] MILOR, L.S., "A tutorial introduction to research on analog and mixed-signal circuit testing," *IEEE Trans. Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 45, no. 10, pp. 1389-1407, oct. 1998.

[42] MORNEAU, M., KHOUAS, A., "Analysis of DC simulation convergence of nonlinear analog circuits with initial solution," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, 2005, pp. 708-712.

[43] NAGI, N., ABRAHAM, J.A., "Hierarchical fault modeling for analog and mixed-signal circuits," in *Proc. IEEE VLSI Test Symp*osium, 1992, pp. 96-101.

[44] NICHOLS, K.G., KAZMIERSKI, T.J., ZWOLINSKI, M., BROWN, A.D., "Overview of SPICE-like circuit simulation algorithms", in *IEE Proc. Circuits, Devices and Systems*, 1994, vol. 141, no. 4, pp. 242-250.

[45] NYE, W., RILEY, D.C., VINCENTELLI, A.S., "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 4, pp. 501-519, avril 1988.

[46] O'LEARY, M., LYDEN, C., "Parametric yield prediction of complex, mixed-signal IC's," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 193-202, mars 1995.

[47] PAPAKOSTAS, D.K., KOSMIDIS, V.C., HATZOPOULOS, A.A., "Analog fault detectability based on statistical circuit analysis," in *Proc. 3th International Conference on Electronics, Circuits and Systems*, 1996, vol. 2, pp. 1076-1079.

[48] PHELPS, R., KRASNICKI, M., RUTENBAR, R.A., CARLEY, L.R., HELLUMS, J.R., "Anaconda: Simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*", vol. 19, no. 6, pp. 703-717, juin 2000.

[49] QUARLES, T., NEWTON, A.R., PEDERSON, D.O., VINCENTELLI, A.S., *SPICE3f User's Manual*, University of California at Berkeley, Californie, États-Unis, mai 1993. [En ligne] http://bear.ces.cwru.edu/eecs_cad/cad_spice.html (Consulté le 27 octobre 2006).

[50] RIFFLET, J.M., YUNES, J.B., *UNIX: Programmation et communication*, Dunod, France, 2003.

[51] SAAB, K., HAMIDA, N.B., KAMINSKA, B., "Closing the gap between analog and digital testing," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 307-314, fév. 2001.

[52] SACHDEV, M., ATZEMA, B., "Industrial relevance of analog IFA: a fact or a fiction », in *Proc. International Test Conference*, 1995, pp. 61-70.

[53] SAVIR, J., GUO, Z., "Test limitations of parametric faults in analog circuits," *IEEE Trans. Instrumentation and Measurement*, vol. 52, no. 5, pp. 1444-1454, oct. 2003.

[54] SENGUPTA, M., SAXENA, S., DALDOSS, L., KRAMER, G., MINEHANE, S., CHENG, J., "Application-specific worst case corners using response surfaces and statistical models," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*", vol. 24, no. 9, pp. 1372-1380, sept. 2005.

[55] SPINKS, S., BELL, I., "ANTICS analogue fault simulation software," in *Proc. IEE Colloquium on Testing Mixed-Signals Circuits and Systems*, 1997, pp. 13/1-13/5.

[56] SPINKS, S.J., CHALK, C.D., BELL, I.M., ZWOLINSKI, M., "Generation and verification of tests for analog circuits subject to process parameter deviations," *Journal of Electronic Testing: Theory and Applications*, vol. 20, no. 1, pp. 11-23, fév. 2004.

[57] SU, H., MICHAEL, C., ISMAIL, M., "Yield optimization of analog MOS integrated circuits including transistor mismatch," *IEEE International Symposium on Circuits and Systems*, 1993, vol. 3, pp. 1804-1804.

[58] TIAN, M.W., SHI, C.-J.R., "Efficient DC fault simulation of nonlinear analog circuits," in *Proc. Design, Automation and Test in Europe*, 1998, pp. 899-904.

[59] TIAN, M.W., SHI, C.-J.R., "Nonlinear analog DC fault simulation by one-step relaxation," in *Proc. 16th IEEE VLSI Test Symposium*, 1998, pp. 126-131.

[60] VAN DER PLAS, G., DEBYSER, G., LEYN, F., LAMPAERT. K., VANDENBUSSHE, J., GIELEN, G.G.E., SANSEN, W., VESELINOVIC, P., LEENAERTS, D., "AMGIE – A synthesis environment for CMOS analog integrated circuits", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1037-1058, sept. 2001.

[61] VLACH, J., SINGHAL, K., *Computer methods for circuit analysis and design*, Van Nostrand Reinhold Company, États-Unis, 1983.

[62] VLADIMIRESCU, A., "SPICE – The fourth decade analog and mixed-signal simulation – a state of the art," in *Proc. International Semiconductor Conference*, 1999, vol. 1, pp. 39-44.

[63] VARIYAM, P.N., CHATTERJEE, A., "FLYER: Fast fault simulation of linear analog circuits using polynomial waveform and perturbed state representation," in *Proc. 10th International Conference on VLSI Design*, 1997, pp. 408-412.

[64] VARIYAM, P.N., CHERUBAL, S., CHATTERJEE, A., "Prediction of analog performance parameters using fast transient testing," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 3, pp. 349-361, mars 2002.

[65] VARIYAM, P.N., HOU, J., CHATTERJEE, A., "Test generation for analog circuits using partial numerical simulation," in *Proc. 12th International Conference on VLSI Design*, 1999, pp. 597-602.

[66] WANG, Z., DIRECTOR, S.W., "An efficient yield optimization method using a two step linear approximation of circuit performance," in *Proc. European Design and Test Conference*, 1994, pp. 567-571.

[67] WONG, M.W.T., WORSMAN, M., "DC nonlinear circuit fault simulation with large change sensitivity," in *Proc. 7th Asian Test Symposium*, 1998, pp. 366-371.

[68] ZWOLINSKI, M., "Relaxation methods for analogue fault simulation," in *Proc. 20th International Conference on Microelectronics*, 1995, vol. 2, pp. 467-471.

[69] ZWOLINSKI, M., CHALK, C., WILKINS, B.R., "Analogue fault modeling and simulation for supply current monitoring," in *Proc. European Design and Test Conference*, 1996, pp. 547-552.

ANNEXE A

Notions de base en simulation analogique

A.1 Introduction

Cette annexe se veut une introduction aux algorithmes utilisés par les simulateurs analogiques de type SPICE. Ces notions sont utiles pour une bonne compréhension du contenu des divers chapitres de ce mémoire.

Tout d'abord, la construction du système matriciel à partir des éléments de circuit en utilisant la notation MNA est abordée. Par la suite, l'algorithme itératif NR appliqué à la résolution des systèmes d'équations non linéaires des circuits y est présenté. Une attention est portée sur ses propriétés de convergence et sur les tests permettant d'arrêter les itérations dans un simulateur SPICE. Enfin, les analyses DC, AC et transitoire sont abordées, avec leur principales caractéristiques.

A.2 Analyse DC des circuits linéaires

L'analyse DC permet de connaître les tensions à chaque nœud du circuit et ce, lorsque celui-ci n'est alimenté que par des tensions continues. Cette analyse est utilisée comme point de départ des autres analyses, comme il sera démontré à la section A.4. Tout d'abord, les circuits les plus simples à résoudre sont ceux qui contiennent uniquement des éléments linéaires, tels des résistances, des condensateurs et des sources de tension ou courant linéaires. Les relations courant / tension demeurent constantes en tout temps pour ces éléments.

A.2.1 Notation sous forme MNA

Le circuit est habituellement représenté sous forme matricielle à l'aide de la loi des nœuds : la somme des courants entrant dans un nœud est égale à la somme des courants en sortant. Ainsi, le vecteur de tensions de nœuds peut être calculé à partir du vecteur des sources de courant constantes et de la matrice de conductances entre chaque nœud. On obtient donc le système matriciel suivant :

$$\boldsymbol{GV} = \boldsymbol{I} \qquad (A.1)$$

Dans cette équation, $\boldsymbol{I}$ est le vecteur des sources de courant constantes, $\boldsymbol{G}$ est la matrice des conductances entre chaque nœud et $\boldsymbol{V}$ est le vecteur des tensions de nœuds recherché. La taille du système est égale au nombre de nœuds n du circuit. Afin de représenter les sources de tension, les amplificateurs idéaux, et tout autre élément produisant une tension, la notation MNA est utilisée. Pour chacun de ces nouveaux éléments, une ligne est ajoutée au système et une nouvelle colonne est ajoutée à la matrice $\boldsymbol{G}$, permettant de modéliser leur relation courant / tension. Pour m sources de tension, on obtient donc un système $(n+m)*(n+m)$.

A.2.2 Gabarits des éléments de circuit

Lors de la construction du système matriciel à partir du circuit, à chaque élément de circuit est associé un gabarit qui permet de l'ajouter au système matriciel de façon transparente. Par exemple, on retrouve fréquemment les gabarits suivants :

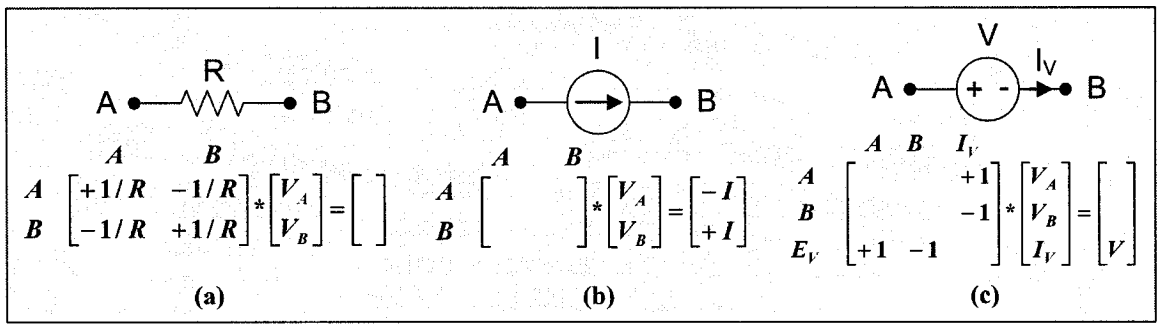


Figure A.1 : Gabarits d'éléments linéaires dans le système MNA : (a) résistance; (b) source de courant indépendante; (c) source de tension indépendante

Une case vide signifie que le composant n'a aucun effet sur cet élément du système MNA. Dans le cas de la source de tension indépendante, une nouvelle ligne contenant l'équation E_V et une nouvelle colonne I_V définissant le courant de source ont été ajoutés. Une liste exhaustive des gabarits des éléments linéaires peut être retrouvée dans [61], page 118. Une fois un composant lu dans le *netlist*, le gabarit correspondant peut alors être créé et ajouté au système.

A.2.3 Résolution du système matriciel

Une fois le système matriciel du circuit construit en utilisant les gabarits des composants, on peut par la suite le résoudre. L'algorithme de *Crout* [14] utilisant la décomposition LU et des substitutions avant et arrière est le plus utilisé puisqu'il ne nécessite aucun espace mémoire supplémentaire pour la résolution. On obtient alors les tensions de nœuds ainsi que les courants circulant dans les sources de tension.

A.3 Analyse DC des circuits non linéaires

Dans le cas où des composants non linéaires, comme des transistors ou des diodes, sont présents dans le circuit, les relations tension / courant entre les terminaux de ces

composants dépendent des tensions entre ces mêmes terminaux, qui sont inconnues. L'algorithme itératif NR est habituellement utilisé pour résoudre de tels circuits.

A.3.1 Algorithme *Newton-Raphson* pour une équation

Le cas particulier de la résolution d'une seule équation non linéaire permet d'illustrer graphiquement le comportement de l'algorithme NR. On considère tout d'abord l'équation non linéaire à résoudre exprimée de la manière suivante :

$$f(x) = 0 \qquad (A.2)$$

Considérant x^0 une approximation initiale de la solution exacte et k l'itération courante, l'algorithme NR nous donne la solution x^k à partir de la solution de l'itération précédente x^{k-1} [14] :

$$x^k = x^{k-1} - \frac{f(x^{k-1})}{f'(x^{k-1})} \qquad (A.3)$$

Chaque itération démarre avec une solution approximative x^{k-1} et recherche une nouvelle approximation x^k, qui pourra être utilisée par l'itération suivante. Les itérations sont poursuivies jusqu'à ce que la convergence soit atteinte. Ceci se produit lorsque la solution entre deux itérations consécutives est identique, c'est-à-dire que $x^k \approx x^{k-1}$. Dans ce cas, $f(x^k) \approx 0$, et la solution approximative x^k devient la solution finale, qui est considérée comme la solution exacte de l'équation non linéaire.

La Figure A.2 illustre le comportement de l'algorithme NR pour une fonction $f(x)$ pour laquelle $f(r) = 0$ à une racine r (solution exacte). Les approximations de la racine r sont respectivement x_0, x_1 et x_2. Pour une approximation donnée x_k, on trouve la dérivée à ce point et on y trace la tangente à $(x_k, f(x_k))$. La nouvelle approximation de la racine, x_{k+1}, se

trouve à l'endroit où cette tangente rencontre l'axe des abscisses, généralement plus près de la solution exacte.

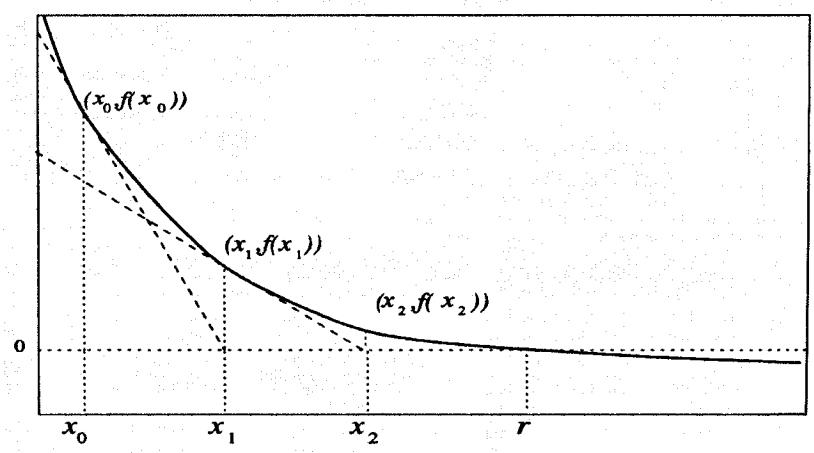


Figure A.2 : Convergence de l'algorithme NR pour une équation (tirée de : [14])

L'erreur e^k associée à l'approximation x^k est définie comme la différence entre celle-ci et sa racine r (solution exacte), c'est-à-dire $e^k = |x^k - r|$. D'après [14], l'erreur e^k à une itération donnée évolue selon l'équation A.4 au voisinage de la racine r :

$$\frac{e^{k+1}}{(e^k)^2} \approx constante \Big|_{x^k = r} \qquad (A.4)$$

La convergence est quadratique lorsqu'on est près de la racine. Par contre, si x^k est très différent de la racine r, il est possible que l'algorithme ne converge jamais.

A.3.2 Algorithme *Newton-Raphson* pour un système d'équations

L'algorithme NR peut être généralisé à un système d'équations non linéaires, ce qui permet son utilisation pour résoudre des circuits électriques comportant des éléments non linéaires. Dans ce cas, la solution est maintenant un vecteur $\boldsymbol{x}$ au lieu d'un scalaire. L'algorithme démarre avec la l'approximation initiale de la solution $\boldsymbol{x}^0$. À chaque

itération k, il utilise la solution approximative précédente x^{k-1} et recherche une nouvelle solution x^k, comme l'illustre l'équation A.5 :

$$x^k = x^{k-1} - \frac{F(x^{k-1})}{J(x^{k-1})} \qquad (A.5)$$

$F(x^{k-1})$ est le vecteur de fonctions à minimiser, alors que $J(x^{k-1})$ est le Jacobien de $F(x^{k-1})$, c'est-à-dire la matrice des dérivées partielles. Toutefois, puisque l'inverse d'une matrice est généralement trop coûteux à calculer, l'équation A.5 est plutôt représentée selon l'équation A.6 ou, de façon plus explicite, selon A.7 :

$$J(x^{k-1})\left(x^k - x^{k-1}\right) = -F(x^{k-1}) \qquad (A.6)$$

$$\begin{bmatrix} \frac{\partial f(x_1{}^{k-1})}{\partial x_1{}^{k-1}} & \frac{\partial f(x_1{}^{k-1})}{\partial x_2{}^{k-1}} & \cdots & \frac{\partial f(x_1{}^{k-1})}{\partial x_n{}^{k-1}} \\ \frac{\partial f(x_2{}^{k-1})}{\partial x_1{}^{k-1}} & \frac{\partial f(x_2{}^{k-1})}{\partial x_2{}^{k-1}} & \cdots & \frac{\partial f(x_2{}^{k-1})}{\partial x_n{}^{k-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(x_n{}^{k-1})}{\partial x_1{}^{k-1}} & \frac{\partial f(x_n{}^{k-1})}{\partial x_2{}^{k-1}} & & \frac{\partial f(x_n{}^{k-1})}{\partial x_n{}^{k-1}} \end{bmatrix} \begin{bmatrix} x_1{}^k - x_1{}^{k-1} \\ x_2{}^1 - x_2{}^{k-1} \\ \vdots \\ x_n{}^k - x_n{}^{k-1} \end{bmatrix} = \begin{bmatrix} f_1{}^{k-1} \\ f_2{}^{k-1} \\ \vdots \\ f_n{}^{k-1} \end{bmatrix} \qquad (A.7)$$

Lorsqu'on est près de la solution exacte, la convergence est quadratique comme c'était le cas pour le cas d'une seule équation, c'est-à-dire que, si x est la solution exacte :

$$\frac{\| x^k - x \|}{\| x^{k-1} - x \|^2} \approx \textit{constante} \qquad (A.8)$$

Toutefois, selon [14], la convergence quadratique est perdue si la matrice jacobienne est singulière à la solution x, c'est-à-dire que son déterminant est nul.

A.3.3 Application en simulation de circuits

Afin d'être en mesure de représenter les composants non linéaires dans le système MNA, ceux-ci doivent d'abord être linéarisés autour d'une approximation de la solution exacte. Cette opération permet d'obtenir une relation tension/courant entre les terminaux de ces composants. La Figure A.3 (a) illustre la linéarisation d'une diode ayant une tension V_d entre ses bornes. On obtient une conductance G_d et un courant équivalent I_{eq}, qui sont exacts uniquement à la tension $V_d = V_B$.

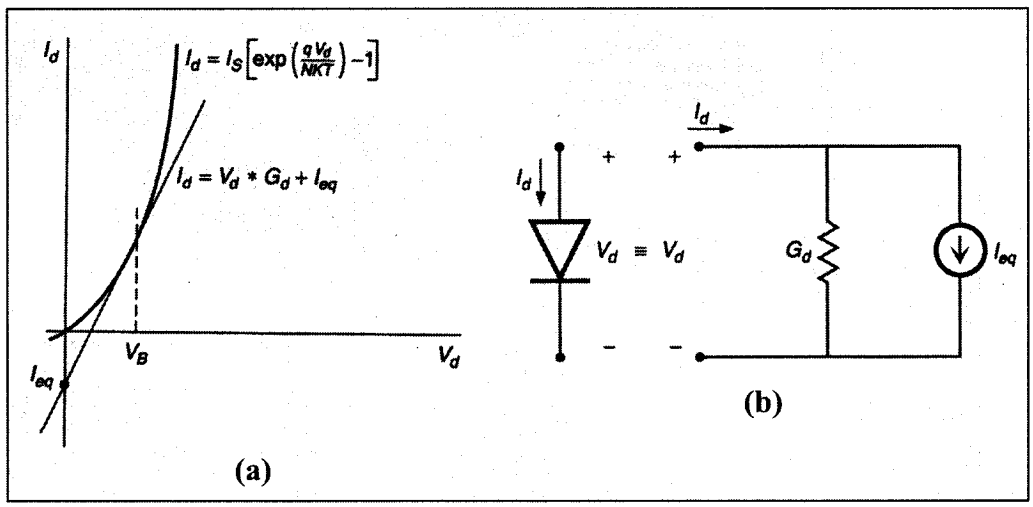
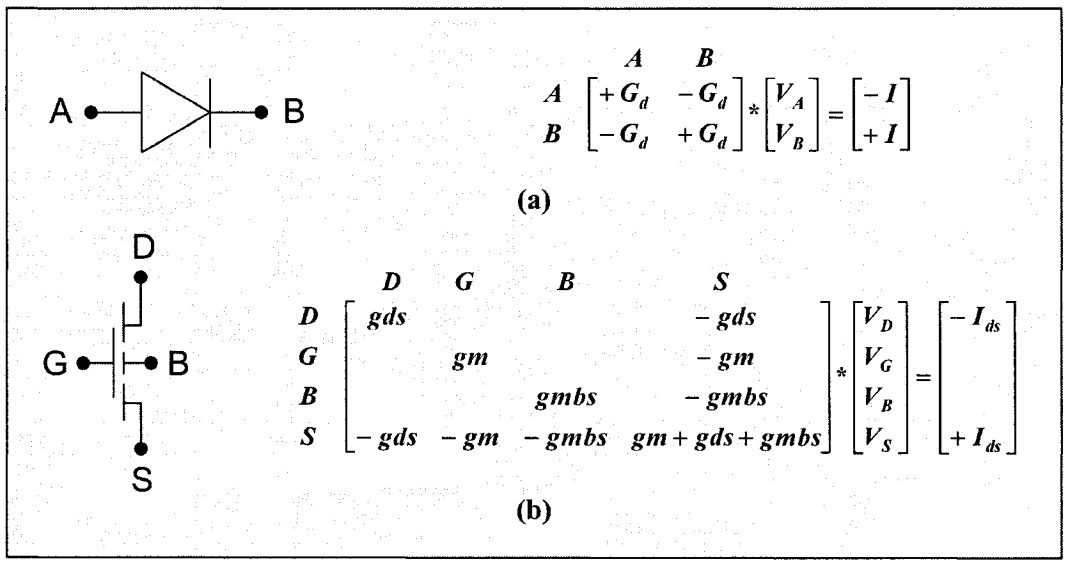


Figure A.3 : Exemple de linéarisation d'une diode (tirée de : [29])
(a) linéarisation autour de V_B; (b) model linéarisé équivalent

La conductance G_d est, en réalité, la dérivée partielle de l'équation de la diode par rapport à la tension V_B, c'est-à-dire δI_d / δV_d lorsque $V_d = V_B$. De même, I_{eq} est une source de courant indépendante qui n'est définie qu'à la tension $V_d = V_B$. Il est possible de transformer n'importe quel composant non linéaire en une série d'éléments linéaires valides pour une solution donnée tels que sur la Figure A.3 (b). Ainsi, à chaque composant non linéaire est associé un gabarit. Les gabarits d'une diode et d'un transistor MOSFET sont représentés à la Figure A.4.



$$\begin{array}{c} \\ A \\ B \end{array}\begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{bmatrix} +G_d & -G_d \\ -G_d & +G_d \end{bmatrix} \end{array} * \begin{bmatrix} V_A \\ V_B \end{bmatrix} = \begin{bmatrix} -I \\ +I \end{bmatrix}$$

$$\begin{array}{c} \\ D \\ G \\ B \\ S \end{array}\begin{array}{c} \begin{array}{cccc} D & G & B & S \end{array} \\ \begin{bmatrix} gds & & & -gds \\ & gm & & -gm \\ & & gmbs & -gmbs \\ -gds & -gm & -gmbs & gm+gds+gmbs \end{bmatrix} \end{array} * \begin{bmatrix} V_D \\ V_G \\ V_B \\ V_S \end{bmatrix} = \begin{bmatrix} -I_{ds} \\ \\ \\ +I_{ds} \end{bmatrix}$$

Figure A.4 : Éléments non linéaires dans le système MNA (tirée de : [29])
(a) diode; (b) transistor à 4 terminaux

Les simulateurs commerciaux possèdent des modèles de composants beaucoup plus complexes qu'illustré sur la Figure A.4, mais le principe demeure le même. La linéarisation des composants non linéaires permet d'obtenir des relations tension / courant qui peuvent être entrées dans le système matriciel MNA.

L'algorithme NR peut être adapté au système matriciel MNA, qui permet maintenant de représenter tant les éléments linéaires que non linéaires. En effet, si on réarrange l'équation A.6 en déplaçant le vecteur d'approximation de la solution, x^{k-1}, du côté droit, on obtient A.9. En remplaçant les termes du cotés droit par $I(x^{k-1})$, on obtient finalement l'équation A.10 :

$$J(x^{k-1})\,x^k = J(x^{k-1})\,x^{k-1} - F(x^{k-1}) \qquad (A.9)$$

$$J(x^{k-1})\,x^k = I(x^{k-1}) \qquad (A.10)$$

L'équation A.10 est similaire à A.1, qui est à la base du système MNA, alors que $\boldsymbol{G} = J(x^{k-1})$, $\boldsymbol{I} = I(x^{k-1})$ et $\boldsymbol{V} = x^k$. Dans ce cas, la matrice de conductances $\boldsymbol{G}$ et le vecteur de

sources indépendantes $\boldsymbol{I}$ sont dépendants de l'approximation de la solution $\boldsymbol{x}^{k-1}$ qui, on l'espère, se rapproche de la solution exacte $\boldsymbol{x}$. Il est nécessaire de recalculer toutes les entrées de la matrice $\boldsymbol{G} = \boldsymbol{J}(\boldsymbol{x}^{k-1})$ et du vecteur $\boldsymbol{I} = \boldsymbol{I}(\boldsymbol{x}^{k})$ à chaque itération k, puisque ces valeurs sont dépendantes du point de polarisation courant $\boldsymbol{x}^{k-1}$. Par la suite, le système matriciel obtenu $\boldsymbol{GV} = \boldsymbol{I}$ peut être résolu pour obtenir une nouvelle approximation de la solution $\boldsymbol{x}^{k}$. À la convergence, lorsque $\boldsymbol{x}^{k} \approx \boldsymbol{x}^{k-1}$, on considère que $\boldsymbol{J}(\boldsymbol{x}^{k}) \approx \boldsymbol{J}(\boldsymbol{x}^{k-1})$ et $\boldsymbol{I}(\boldsymbol{x}^{k}) \approx \boldsymbol{I}(\boldsymbol{x}^{k-1})$. Dans ce cas, la solution finale $\boldsymbol{x}^{k}$ est considérée comme la solution exacte du circuit non linéaire pour le reste de ce mémoire.

A.3.4 Critère d'arrêt des itérations

À chaque itération, un critère d'arrêt est utilisé afin de déterminer si la convergence est atteinte ou si l'algorithme doit poursuivre les itérations. Dans les simulateurs SPICE standard, un critère semblable à celui des équations A.11 et A.12 est utilisé. À la fin de l'itération k, une tolérance tol_i^k est définie pour la valeur d'indice i du vecteur de solution, qui peut être une tension de nœud ou un courant de branche d'après la notation MNA. Cette tolérance est représentée à l'équation A.11. La variation de cette valeur entre deux itérations doit être inférieure à sa valeur de tolérance tol_i^k pour que la convergence soit atteinte. Si au moins une valeur x_i^k de la solution varie par plus que sa valeur de tolérance tol_i^k, l'algorithme NR continue les itérations.

$$\boldsymbol{tol}_i^{\,k} = \boldsymbol{abstol} + \boldsymbol{reltol} \times \max\left(|\, x_i^{\,k} \,|, |\, x_i^{\,k-1} \,|\right) \qquad (A.11)$$

$$\text{Si} \quad |\, x_i^{\,k} - x_i^{\,k-1} \,| \leq \boldsymbol{tol}_i^{\,k} \quad \forall\, \boldsymbol{i} \quad \rightarrow \text{convergence atteinte} \qquad (A.12)$$

Les valeurs *abstol* et *reltol* sont des paramètres standard des simulateurs SPICE, représentant respectivement une tolérance absolue et une tolérance relative. Les valeurs par défaut dans *SPICE3f5* et *Spectre* sont : *abstol* = 10^{-6} V pour les tensions ou 10^{-9} A pour les courants, alors que *reltol* = 0.001. À moins d'avoir des valeurs de l'ordre de *abstol*, l'influence de *abstol* est négligeable par rapport à *reltol*. Lorsque l'algorithme NR

converge selon ce critère, la solution finale x^k est considérée comme la solution exacte générée par le simulateur.

A.3.5 Complexité de la simulation DC

L'algorithme NR, tel qu'implanté dans les simulateurs SPICE, est présenté à la Figure A.5.

```
1   Étant donnée une approximation x^0 de la solution
2   Étant donnés un nombre maximal d'itérations N et une tolérance ε
3   k = 0
4   Tant que k < N
5       k = k+1  // incrémentation du nombre d'itérations
6       Calculer J(x^(k-1)) et I(x^(k-1)) en linéarisant les composants autour de x^(k-1)
7       Résoudre le système J(x^(k-1)) x^k = I(x^(k-1)) par l'algorithme de Crout
8       Appliquer le critère d'arrêt (éq. A.11 et A.12)
9       Si convergence atteinte
10          La solution finale x^k est retournée comme étant la solution exacte
11          Terminer l'algorithme
12      Fin Si
13  Fin Tant que
```

Figure A.5 : Algorithme NR pour la résolution de circuits non linéaires

Dans l'algorithme de la Figure A.5, le temps de calcul est presque exclusivement consommé par deux opérations : le calcul de $\boldsymbol{J}(\boldsymbol{x}^{k-1})$ et $\boldsymbol{I}(\boldsymbol{x}^{k-1})$ à partir des modèle des composants (ligne 6) et la résolution du système d'équations linéaires (ligne 7).

Évaluation des modèles

Puisque la solution $\boldsymbol{x}^k$ change à chaque itération k, les modèles doivent être évalués à chaque itération pour obtenir le modèle linéarisé qui est inséré dans le système MNA. Les

composants linéaires sont très rapides à évaluer puisqu'ils sont constants peu importe la polarisation DC. Par contre, les composants non linéaires sont beaucoup plus complexes à évaluer et dépendent de nombreux paramètres. Les modèles existant de nos jours sont extrêmement précis mais nécessitent un temps de calcul important. Ceci est particulièrement notable au niveau des transistors MOS, qui dépendent de plusieurs dizaines de paramètres différents. Quelques dizaines d'équations doivent donc être calculés pour chaque instance dans le circuit, dépendant de la complexité du modèle. Le temps nécessité par l'évaluation des composants est donc proportionnel au nombre de composants non linéaires retrouvés dans le circuit, particulièrement les transistors MOS.

Résolution du système linéaire

Cette opération consiste essentiellement à effectuer l'algorithme de *Crout* pour résoudre un système d'équations linéaires. Sa complexité en temps de calcul est de l'ordre de $\theta(n^3)$ pour un système de taille n [14], ce qui est prohibitif pour des circuits de grande taille. Par contre, les matrices jacobiennes sont généralement creuses (*sparse*), c'est-à-dire que la grande majorité des entrées ont une valeur nulle. En effet, un nœud donné du circuit n'est pas relié à tous les autres nœuds, mais seulement à quelques voisins. Cette propriété est exploitée en tenant compte uniquement des valeurs non nulles contenues dans la matrice. L'utilisation de telles méthodes permet d'obtenir expérimentalement une complexité de l'ordre de $\theta(n^{1.5})$ [22], ce qui est beaucoup plus acceptable.

Complexité globale

Des deux sous-sections précédentes, le coût en temps de calcul est donné par l'équation suivante [22] :

$$Coût = N\left(K_1 t + K_2 n^{1.5}\right) \qquad (A.13)$$

Dans l'éq. A.13, N est le nombre d'itérations NR et t est le nombre de composants non linéaires. En pratique, K_1 est beaucoup plus élevé que K_2 pour un circuit composé de

transistors. C'est pourquoi, pour de petits circuits, le temps de calcul dépend principalement de l'évaluation des modèles, alors que pour un circuit de grande taille, $n^{1.5}$ devient important par rapport à t et le temps de calcul dépend surtout de la résolution du système matriciel. Dans les deux cas, par contre, le temps de calcul est proportionnel au nombre d'itérations.

A.3.6 Problèmes de convergence

Afin de trouver le point de polarisation d'un circuit, l'algorithme NR nécessite généralement entre 10 et 100 itérations. Si, après un certain nombre d'itérations la convergence n'est pas atteinte, les simulateurs commerciaux font appel à des techniques dites de continuation afin de trouver la solution au terme d'un certain nombre d'étapes intermédiaires. Ces techniques exécutent l'algorithme NR plusieurs fois, en commençant par une version du circuit généralement plus facile à résoudre, puis en augmentant la complexité jusqu'au circuit nominal [39]. La première, appelée *source stepping*, consiste à augmenter graduellement les sources, de zéro à leur valeur nominale. La solution à chaque étape est utilisée pour approximer la solution à l'étape suivante, jusqu'à la dernière, qui donne la solution exacte recherchée. Cette méthode s'appuie sur le fait que le circuit est généralement plus simple à résoudre lorsque ses sources sont très faibles. Une autre technique similaire, *gmin stepping*, consiste à imposer une conductance minimale entre chacun des nœuds reliés par un composant, permettant d'éliminer les circuits ouverts qui peuvent causer des problèmes d'imprécision numérique. Cette fois, c'est la conductance minimale qui est variée, d'une valeur élevée à une valeur faible (circuit nominal). D'autres techniques de continuation existent, mais aucune ne garantit une solution. En effet, à chaque étape, il y a un risque de non convergence, qui fait avorter le processus et aucun résultat n'est obtenu. Heureusement, les simulateurs commerciaux sont sophistiqués et robustes, ce qui fait en sorte que les problèmes de convergence sont très rares de nos jours.

A.4 Types d'analyse

Jusqu'ici, nous avons décrit de quelle façon il est possible de trouver le point de polarisation DC d'un circuit non linéaire. Les autres types d'analyse de circuit sont réalisés de façon similaire.

A.4.1 Analyse balayage DC (DC sweep)

Cette analyse recherche la fonction de transfert DC des sorties par rapport à un paramètre spécifié. Il peut s'agir de la température, d'une source indépendante ou d'un autre paramètre pour lequel un intervalle de variation et un pas sont définis. Un simulateur considère une analyse DC comme une série de points d'opération DC effectués de façon séquentielle pour chaque valeur attribuée au paramètre. Toutefois, la solution pour une valeur donnée du paramètre varié a toutes les chances d'être semblable à celle de la valeur précédente. Or, comme démontré à la section A.3, l'algorithme NR converge rapidement lorsqu'il est près de la solution exacte. C'est pourquoi la solution du point précédent est habituellement choisie comme première approximation $\boldsymbol{x}^0$ de la solution du point suivant.

A.4.2 Analyse AC

L'analyse AC consiste à déterminer la réponse en fréquence du circuit en mode petit signal. Tout d'abord, le point de polarisation DC est calculé à l'aide de l'algorithme NR. Par la suite, les éléments non linéaires sont remplacés par leur équivalent petit signal au point de polarisation. Les effets non linéaires, tels la distorsion et la saturation sont donc négligés. L'analyse AC est par la suite réalisée en effectuant une série de résolution du système linéaire $\boldsymbol{YV} = \boldsymbol{I}$, où $\boldsymbol{Y}$ est la matrice d'admittance. Cette fois, les réactances des composants qui emmagasinent de l'énergie, comme les condensateurs et les inductances, sont considérées. Pour chaque fréquence f à simuler, $\omega = 2\pi f$ et on obtient un nouveau

système à résoudre puisque la composante complexe de la matrice ***Y*** est modifiée. L'algorithme de *Crout* est exécuté une fois pour chaque fréquence. Le temps de simulation AC dépend d'une part du nombre d'itérations nécessaires pour trouver le point de polarisation et d'une autre part du nombre de fréquences à analyser.

A.4.3 Analyse transitoire

L'analyse transitoire est la plus complexe et habituellement la plus longue à réaliser. Comme pour l'analyse AC, elle débute par le calcul du point de polarisation DC. On obtient alors la solution au temps 0 s. À chaque intervalle de temps, une intégration numérique permet de mettre à jour la quantité de charges emmagasinée par les capacités et inductances. Par la suite, un nouveau point de polarisation est calculé avec l'algorithme NR, en considérant les changements de tension / courant aux bornes des condensateurs et inductances causés par les échanges d'énergie. Les intervalles de temps sont définis de façon dynamique selon les options offertes par le simulateur. Des intervalles plus courts augmentent le temps de calculs, mais sont nécessaires pour assurer la précision lorsque des variations rapides de tensions sont rencontrées. Plus de détails sur l'analyse transitoire peuvent être obtenus dans [7], [29] et [61]. Il est important de noter que, comme c'est le cas pour l'analyse de balayage DC, la solution du point précédent est utilisée comme approximation de la solution au point courant et ce, afin de réduire le nombre d'itérations NR nécessaires.

ANNEXE B

Description du simulateur *MultiSPICE*

Cette annexe décrit de façon détaillée le fonctionnement du simulateur *MultiSPICE* développé à partir de *SPICE3f5*. Nous y aborderons tout d'abord le fonctionnement général, puis un guide d'utilisation sera présenté.

B.1 Fonctionnement général

Le simulateur proposé consiste en une série de fonctionnalités qui ont été ajoutées au simulateur existant *SPICE3f5*, développé à l'université de Californie à Berkeley. Le logiciel original est programmé en langage C et le code source est disponible dans [49]. La figure suivante illustre le schéma interne du simulateur *MultiSPICE* proposé.

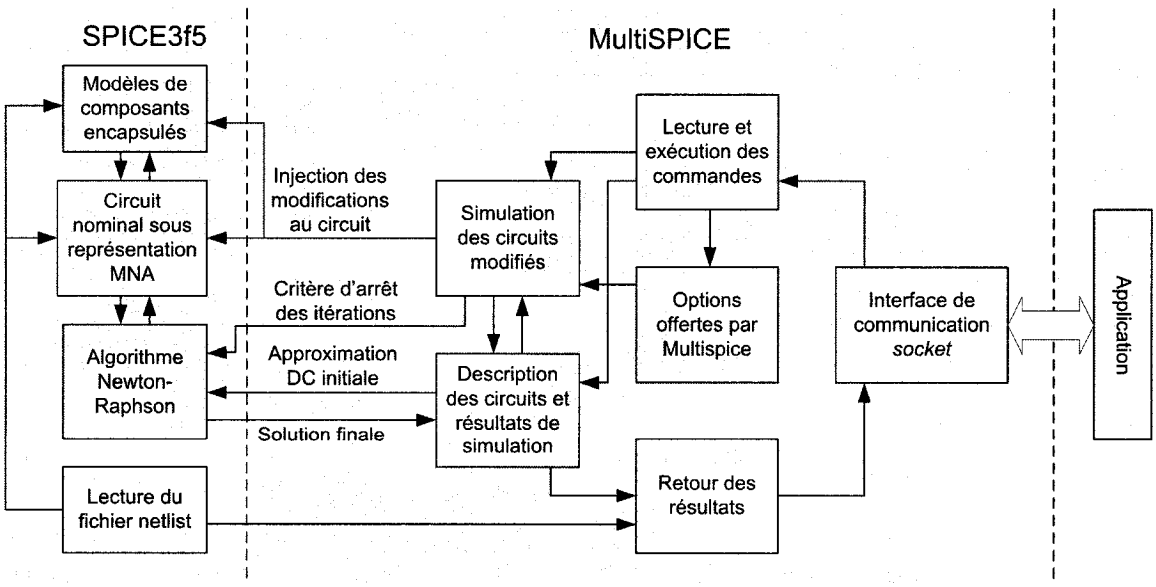


Figure B.1 : Schéma interne du simulateur *MultiSPICE*

Les fonctionnalités proposées par *MultiSPICE* nécessitaient le développement d'une interface de communication *socket*, d'un analyseur syntaxique, d'une liste d'options de simulation additionnelles, d'une structure de données efficace ainsi que d'un module effectuant les simulations séquentiellement. L'implémentation a été réalisée de façon à ce que le fonctionnement normal du logiciel original ne soit pas affecté. Les nouvelles fonctionnalités proposées ont ainsi été encapsulées dans de nouvelles structures, qui ont été greffées aux structures du logiciel existant. Par conséquent, les modifications effectuées aux fichiers sources existants sont mineures. Des fonctions permettent d'injecter des modifications à l'intérieur de la représentation MNA du circuit initial et dans les modèles de composants, de spécifier l'AIPS à utiliser et le critère d'arrêt des itérations avant la convergence. Une fois la simulation de chaque circuit terminée, les résultats sont récupérés pour être retournés à l'application.

De façon générale, des nœuds de sortie ainsi que des options de simulation sont spécifiés au simulateur *MultiSPICE*. Par la suite, un groupe de circuits (*batch*) est créé. Chaque circuit est décrit par une série de modifications à effectuer sur le circuit original. Des analyses de point de polarisation DC, balayage DC, AC et transitoire peuvent ensuite être effectuées sur chaque circuit du groupe. Les résultats sont conservés pour chaque point de simulation afin d'être retournés et parfois réutilisés dans le calcul d'une AIPS ultérieure. Dès que les résultats de simulation sont retournés et ne seront plus utilisés, ils peuvent être effacés afin de libérer de la mémoire. Des messages d'erreurs / avertissement ayant trait à des mauvaises commandes reçues ou à des problèmes de convergence sont également renvoyés à l'application.

Afin de permettre une analyse plus approfondie du comportement à la convergence, il est possible de conserver les résultats intermédiaires à chaque itération NR en utilisant l'option *saveiter*. Ainsi, la solution générée par chaque itération est conservée. Si la convergence n'est pas atteinte après le nombre maximal d'itérations, les méthodes de

continuation sont utilisées. Les itérations ayant trait à chaque point intermédiaire permettant d'obtenir la solution finale désirée sont conservées.

B.2 Guide d'utilisation

Cette section présente un guide d'utilisation du simulateur *MultiSPICE*. Pour une meilleure compréhension, il est fortement recommandé de survoler tout d'abord le guide d'utilisation de *SPICE3f5* [49].

B.2.1 Démarrage du simulateur

Un nouveau mode de démarrage par rapport au simulateur *SPICE3f5* permet d'activer la communication *socket* nécessaire au fonctionnement de *MultiSPICE*. L'option –m active ce mode, qui demande à *MultiSPICE* de se connecter sur la machine qui roule l'application pour recevoir les instructions de simulation. La commande suivante est donc utilisée :

```
<executable> -m <machine.domaine> <port> <taille_tampon>
             [-d <fichier_trace>]
```

L'argument *machine.domaine* désigne la machine sur laquelle roule l'application qui appelle le simulateur. L'argument *port* désigne le numéro de port sur lequel le simulateur doit se connecter. La taille du tampon mémoire utilisé pour l'échange de commandes et de résultats entre le simulateur et l'application est spécifiée par l'argument *taille_tampon*. Les détails des opérations de simulation effectuées sont sauvegardés dans le fichier *fichier_trace* si l'option –d est utilisée. Sinon, ces opérations sont affichées à l'écran.

Par exemple, la commande suivante invoque le simulateur en mode *MultiSPICE*, qui se connecte par la suite sur le port *39480* de la machine *pcak01.grm.polymtl.ca*. Des communications de 10 KO seront échangées avec l'application. Les étapes de simulation sont sauvegardées dans le fichier *trace.out*.

```
spice3 -m pcak01.grm.polymtl.ca 39480 10000 -d trace.out
```

L'application doit alors être en mode d'attente d'une connexion sur ce port. En d'autres mots, elle doit être bloquée sur une commande POSIX *accept*. Elle continue dès que le simulateur s'est connecté sur le port spécifié.

B.2.2 Instructions supportées

Cette section présente les instructions supportées par le simulateur *MultiSPICE*. Ces commandes débutent toutes par « . » et se terminent par un changement de ligne « \n ».

`.source <fichier_netlist>`

Similaire à son équivalent dans *SPICE3f5*, cette commande ouvre et lit le fichier *netlist* spécifié. La représentation MNA du circuit nominal est alors construite et les paramètres des composants du circuit sont initialisés. Il s'agit de la première commande à appeler après le démarrage de *MultiSPICE*, car toute autre commande appelée avant *.source* produira un message d'erreur.

`.gettitle`

Retourne le titre du circuit actuellement en mémoire. Le titre est la première ligne du fichier *netlist* qui a été lu.

`.getcomponents`

Retourne la liste des composants du circuit actuellement en mémoire. Pour chaque composant du circuit, son type, son modèle et son nom tels que définis dans le *netlist* sont retournés.

`.getnodes`

Retourne la liste des nœuds du circuit actuellement en mémoire. Seul le nom de chaque nœud, tel que défini dans le *netlist*, est retourné.

`.getcparam <type_composant> <nom_paramètre>`

Retourne les valeurs données au paramètre spécifié pour chaque instance du type de composant spécifié. Les types de composants ainsi que les paramètres supportés sont les mêmes que *SPICE3f5*. Pour chaque instance du type de composant spécifié, le nom du composant, le nom du paramètre et sa valeur telle que dans le *netlist* sont retournés.

`.getmparam <type_composant> <nom_paramètre>`

Retourne les valeurs données au paramètre spécifié pour chaque modèle du type de composant spécifié. Les types de composants ainsi que les paramètres supportés sont les mêmes que *SPICE3f5*. Pour chaque modèle du type de composant spécifié, le nom du modèle, le nom du paramètre et sa valeur telle que dans le *netlist* sont retournés.

`.setoutput [+ | -] <node_name>`

Le nœud spécifié est considéré comme un nœud de sortie pour le circuit. Si le signe « - » est utilisé, ce nœud n'est plus un nœud de sortie s'il l'était auparavant. Seuls les résultats de simulation correspondant aux nœuds de sortie sont retournés à l'application. De plus, l'algorithme *TBSA* requiert les nœuds de sortie pour terminer les itérations avant la convergence.

`.batch <batch_ID>`

Spécifie le numéro d'identification du groupe de circuits qui sera utilisé. Si le groupe n'existe pas encore, il est créé et les prochains circuits spécifiés seront insérés automatiquement dans ce groupe. Si le groupe existe déjà, il est sélectionné. Aucun nouveau circuit ne pourra y être ajouté. Les prochaines tâches de simulation spécifiées seront effectuées sur les circuits de ce groupe.

`.circuit <circuit_ID> <solution_initiale> <circuit_comparaison>`
`[save | flush]`

Spécifie un nouveau circuit modifié, qui sera identifié par son numéro d'identification (ID). Les numéros d'identification doivent être consécutifs, sinon une erreur est générée. Une expression permettant de calculer l'AIPS pour la simulation DC est ensuite fournie, suivie du numéro d'identification du circuit nominal avec lequel ce circuit doit être comparé pour l'algorithme *TBSA*. Enfin, le mot-clé *save* est utilisé si les résultats de simulation doivent être conservés pour être réutilisés ultérieurement. Sinon, les résultats sont automatiquement effacés dès qu'ils ont été retournés à l'application après la simulation. Les expressions suivantes permettent de définir l'AIPS :

→ $<ID> : Numéro d'identification du circuit dont on utilise la solution finale pour le même point de simulation que le point actuel. Cette valeur correspond à un vecteur.

→ <1-9.> : Nombre réel scalaire par lequel un vecteur peut être multiplié ou divisé.

→ <+,-,*,/,(,)> : Opérateurs arithmétiques.

Exemples :

`$0` : Réutilisation de la solution du circuit initial.

`($1+$2)/2` : Moyenne des solutions des circuits 1 et 2.

`.cparam <nom_composant> <nom_paramètre> <valeur_paramètre>`

Injecte une modification au paramètre spécifié du composant dans le dernier circuit modifié. La nouvelle valeur de ce paramètre est celle indiquée et elle ne sera utilisée que pour ce circuit.

`.mparam <nom_modèle> <nom_paramètre> <valeur_paramètre>`

Injecte une modification au paramètre spécifié du modèle dans le dernier circuit modifié. La nouvelle valeur de ce paramètre est celle indiquée et elle ne sera utilisée que pour ce circuit.

`.short <noeud1> <noeud2> <résistance>`

Injecte un court-circuit entre les deux nœuds spécifiés dans le dernier circuit modifié. Une résistance de la valeur spécifiée sera donc utilisée entre ces nœuds pour ce circuit seulement.

`.cshort <nom_composant> <terminal1> <terminal2> <résistance>`

Injecte un court circuit entre les deux terminaux spécifiés du composant. Une résistance de la valeur spécifiée sera donc utilisée entre ces terminaux pour ce circuit seulement. Chaque terminal est identifié par une seule lettre : +,-,g,d,s,b,c,e…

`.open <nom_composant>`

Force le composant non linéaire spécifié en zone *cut-off* pour le dernier circuit modifié. Le composant est donc déconnecté pour ce circuit seulement. Les composants supportés sont les transistors et les diodes.

`.op`

Effectue une analyse de point de polarisation DC sur tous les circuits du dernier groupe spécifié. Le point de polarisation DC sera donc calculé et les tensions / courants à chaque nœud de sortie de chaque circuit du groupe seront retournés. Cette commande est similaire à celle utilisée dans *SPICE3f5*.

`.dc <nom_source> <valeur_départ> <valeur_finale> <valeur_pas>`

Effectue une analyse de type balayage DC sur tous les circuits du dernier groupe spécifié. Le nombre de points à simuler est égal à la valeur finale moins la valeur de départ, le tout divisé par le pas. Cette commande est similaire à celle utilisée dans *SPICE3f5*. Pour chaque nœud de sortie de chaque circuit du groupe, le résultat à chaque point sera retourné.

`.ac <dec | oct | lin> <nombre_points> <fréq_départ> <fréq_finale>`
Effectue une analyse de type fréquentielle AC sur tous les circuits du dernier groupe spécifié. La fréquence peut évoluer par décade, octave ou linéairement. Cette commande est similaire à celle utilisée dans *SPICE3f5*. Pour chaque nœud de sortie de chaque circuit du groupe, le résultat à chaque valeur de fréquence sera retourné.

`.tran <delta_temps> <temps_final> [<temps_départ> [<delta_max>]]`
Effectue une analyse de type transitoire sur tous les circuits du dernier groupe spécifié. La simulation est effectuée jusqu'au temps spécifié et les résultats sont retournés pour chaque intervalle de temps. Le nombre de points retournés est donc égal au temps final moins le temps de départ, le tout divisé par le delta de temps. Cette commande est similaire à celle utilisée dans *SPICE3f5*. Pour chaque nœud de sortie de chaque circuit du groupe, le résultat à chaque intervalle de temps sera retourné.

`.min <tâche_simulation> <nœud> <nombre_points> <val1> <val2>...`
Spécifie le vecteur contenant le seuil de comparaison minimal ($V_{min,th}$) pour le nœud de sortie spécifié dans le cas où la méthode *TBSA* est utilisée. La taille de ce vecteur est égale au nombre de points DC à simuler. Le seuil maximal est fourni par la commande *.max*.

`.max <tâche_simulation> <nœud> <nombre_points> <val1> <val2>...`
Spécifie le vecteur contenant le seuil de comparaison maximal ($V_{max,th}$) pour le nœud de sortie spécifié dans le cas où la méthode *TBSA* est utilisée. La taille de ce vecteur est égale au nombre de points DC à simuler. Le seuil minimal est fourni par la commande *.min*.

`.option <nom_option_et_valeurs>`
Spécifie une option de simulation. Les options existantes avec *SPICE3f5* sont toujours supportées. Les options supplémentaires supportées par *MultiSPICE* se retrouvent dans le tableau suivant :

Tableau B.1 : Options supportées par *MultiSPICE*

Option	Effet	Valeur par défaut
`multispice`	Active le mode *MultiSPICE*	activé
`spice3`	Désactive le mode *MultiSPICE*	désactivé
`init <0 | 1>`	Utilisation d'une AIPS	1 : activé
`stop < convergence | iteration | reltol | tbsa >`	Critère d'arrêt des itérations NR : arrêt à la convergence, nombre fixe d'itérations, augmentation de *reltol* ou *TBSA*	convergence
`iterstop <nombre>`	Nombre d'itérations NR avant l'arrêt de la simulation DC, lorsque la méthode du nombre fixe d'itérations est sélectionnée	1
`reltol2 <valeur>`	Valeur de tolérance relative à utiliser pour terminer la simulation DC lorsque la méthode d'augmentation de la tolérance relative est sélectionnée	0.001
`tbsa <one | all | half>`	Le critère d'arrêt de la méthode *TBSA* est valide si une, toutes ou la plupart des valeurs de sortie sont suffisamment précises pour être comparées aux seuils fournis	one
`converge <0 | 1>`	Simule jusqu'à la convergence malgré un critère d'arrêt spécifié (utile pour mesurer la précision d'un critère d'arrêt)	0 : désactivé
`stats <0 | 1>`	Génère de l'information et des statistiques sur la simulation dans le fichier *stats/batch_<ID>*	0 : désactivé
`saveiters <0 | 1>`	Sauvegarde les résultats obtenus à chaque itération NR; si *stats* est activé, celles-ci se retrouveront dans le fichier *stats/batch_<ID>*	0 : désactivé
`flushiters <0 | 1>`	Si *saveiters* est activé, efface les résultats de chaque itération une fois que le point de simulation est terminé	0 : activé

`.flush batch <batch_ID>`

Efface de la mémoire toute l'information sur le groupe de circuits spécifié. Tant l'information sur les circuits que les résultats de simulation sont détruits. Ce groupe ne pourra plus être réutilisé à l'avenir.

`.flush circuit <circuit_ID>`

Efface de la mémoire toute l'information sur le circuit spécifié. Tant l'information sur le circuit que ses résultats de simulation sont détruits. Ce circuit ne pourra plus être réutilisé à l'avenir.

`.flush all`

Efface de la mémoire toute l'information sur tous les groupes de circuits. Tant l'information sur les groupes et les circuits que les résultats de simulation sont détruits.

`.flush <op | dc | ac | tran> <all | first | last>`

Détruit les résultats de simulation pour le type d'analyse spécifié dans tous les circuits. Si *all* est sélectionné, tous les résultats pour ce type d'analyse sont détruits. Si *first* est sélectionné, seule la première tâche de ce type d'analyse est effacée pour chaque circuit. Au contraire, *last* efface la dernière tâche de ce type d'analyse.

`.save batch <fichier_sauvegarde> <batch_ID>`

Sauvegarde l'information sur toutes les tâches de simulation effectuées ainsi que sur chacun des circuits du groupe spécifié. Tant l'information sur les circuits que les résultats de simulation sont sauvegardés.

`.save circuit <fichier_sauvegarde> <batch_ID>`

Sauvegarde toute l'information pour le circuit spécifié. Tant les modifications associées à ce circuit que ses résultats de simulation sont sauvegardés.

`.save <fichier_sauvegarde>`

Sauvegarde l'information sur toutes les tâches de simulation effectuées pour chaque groupe ainsi que sur chacun des circuits. Tant l'information sur les circuits que les résultats de simulation sont sauvegardés.

`.load <fichier_chargement>`

Restore l'information précédemment sauvegardée. La structure de donnée est reconstruite telle qu'elle était lors de la sauvegarde.

`.end`

Fin de la liste d'instructions. Aucune opération n'est effectuée par le simulateur avant la lecture de la commande *.end*. Le simulateur attend donc d'autres communications via la connexion, jusqu'à ce que *.end* soit reçu. Dès lors, les tâches de simulation sur le dernier groupe de circuits débutent.

`.quit`

Le simulateur se termine immédiatement de façon normale.

B.2.3 Résultats retournés

Cette section présente le format des résultats retournés par le simulateur *MultiSPICE*. À l'exception des informations sur les composants, paramètres et nœuds du circuit, chaque ligne débute par « . ».

`.title <titre_du_circuit>`

Titre du circuit, renvoyé immédiatement après une commande *gettitle*. Le titre est la première ligne du fichier *netlist* qui a été lu.

```
<type_composant1> <modèle_composant1> <nom_composant1>
<type_composant2> <modèle_composant2> <nom_composant2> ...
.end
```

Liste de tous les composants retrouvés dans le circuit, retournés immédiatement après une commande *getcomponents*. Chaque composant est décrit par son type et son modèle.

```
<noeud1> <noeud2> ...
.end
```

Liste de tous les nœuds retrouvés dans le circuit, retournés immédiatement après une commande *getnodes*.

```
<composant1> <paramètre> <valeur> <composant2> <paramètre>
<valeur> ...
.end
```

Liste des valeurs de paramètre de tous les composants présents dans le circuit pour un type de composant et de paramètre. Retourné après une commande *getcparam*.

```
<modèle1> <paramètre> <valeur> < modèle2> <paramètre> <valeur>...
.end
```

Liste des valeurs de paramètre de tous les modèles présents dans le circuit pour un type de composant et de paramètre de modèle. Retourné après une commande *getmparam*.

```
.batch <batch_ID>
```

Retourne les résultats de simulation pour le groupe de circuits spécifié. Retourné après une série d'instructions de simulation de circuits.

```
.analysis <OP | DC | AC | TRAN>
```

Retourne les résultats pour la tâche de simulation spécifiée et ce, pour le groupe précédemment spécifié.

```
.circuit <circuit_ID>
```

Les prochains résultats de simulation concernent le circuit spécifié.

`.node <node_name> <valeur1> <valeur2> <valeur3>...`

Tension / courant de sortie correspondant au nœud spécifié pour chaque point de simulation de l'analyse courante, dans le circuit modifié courant. Dans le cas de l'analyse AC, il s'agit de la partie réelle des valeurs.

`.imag <valeur1> <valeur2> <valeur3>...`

Dans le cas de l'analyse AC, partie imaginaire de la tension ou du courant de sortie correspondant au dernier nœud spécifié pour chaque point de simulation de l'analyse courante, dans le circuit modifié courant.

`.INFO <message>`

Message d'information à afficher à l'utilisateur, qui se termine à la fin de la ligne.

`.WARNING <message>`

Message d'avertissement à afficher à l'utilisateur, qui se termine à la fin de la ligne.

`.ERROR <message>`

Message d'erreur à afficher à l'utilisateur, qui se termine à la fin de la ligne.

`.cputime <valeur>`

Temps de calcul, en secondes, nécessité au total par la dernière commande de simulation de circuits.

`.iterations <valeur>`

Nombre d'itérations NR nécessitées au total par la dernière commande de simulation de circuits.

`.end`

Tous les résultats de simulation correspondant à la dernière commande de simulation de circuits ont été envoyés. *MultiSPICE* retourne à l'état d'attente.

B.2.4 Mode automatique

Cette section présente quelques instructions qui ont été ajoutés à *MultiSPICE* afin de créer automatiquement des circuits modifiés pour la simulation de pannes, le dimensionnement automatique de circuits et l'analyse Monte Carlo. Ces instructions, qui sont ajoutées dans le fichier *netlist*, génèrent les circuits modifiés requis, puis ceux-ci sont simulés dès que la lecture du *netlist* est terminée. Ces instructions permettent d'évaluer rapidement l'efficacité de critères d'arrêt des itérations avant la convergence sans nécessiter d'application externe. Elles doivent être utilisés avec le mode *batch* de lancement de *SPICE3f5*, c'est-à-dire avec la commande suivante :

```
<executable> -b <fichier_netlist>
```

```
&out <nœud> <abs | rel | minmax> <val1> [<val2>]
```

Le nœud spécifié est considéré comme une sortie, et le type d'intervalle spécifié sera utilisé par la méthode *TBSA* : *nominal* ± *val1* ; *nominal* * (1 ± *val1*) ; *val1* à *val2*.

```
&inject <méthode_initsol> <type_composant> <paramètre> <val1>
        <val2> <val3> ...
```

Injecte une panne paramétrique pour le paramètre spécifié et chaque instance du type de composant spécifié. Une liste de valeurs fautives est ensuite donnée. Un circuit défectueux est généré pour chaque composant et chaque valeur fautive à donner à ce paramètre. La méthode utilisée pour calculer l'AIPS de chaque circuit doit être spécifiée par un numéro :

1 = Circuit sans faute (x_0).

2 = Circuit précédent selon un ordonnancement des valeurs fautives pour chaque composant (x_{i-1}).

3 = Interpolation simple avec les deux circuits précédents ($x_{i-1} + (x_{i-1} - x_{i-2})$).

4 = Interpolation linéaire avec les deux circuits précédents en tenant compte des valeurs fautives ($x_{i-1} + (x_{i-1} - x_{i-2}) * (val_i/val_{i-1}) / (val_{i-1}/val_{i-2})$).

```
&injectm <méthode_initsol> <type_composant> <paramètre> <val1>
         <val2> <val3> ...
```

Injecte une panne paramétrique pour le paramètre spécifié et chaque modèle du type de composant spécifié. Une liste de valeurs fautives est ensuite donnée. Un circuit défectueux est généré pour chaque modèle et chaque valeur fautive à donner à ce paramètre. Les méthodes permettant de calculer les AIPS sont les mêmes que pour la commande précédente.

```
&shorts <type_composant> <terminal1> <terminal2> <résistance>
```

Injecte un court-circuit résistif entre les terminaux spécifiés de chaque instance du type de composant. Un circuit défectueux est généré pour chaque composant du type spécifié dans le circuit.

```
&opens <type_composant> <résistance>
```

Chaque instance du type de composant non linéaire spécifié est forcée en zone de *cut-off* et ses terminaux sont court-circuités par la résistance donnée. Un circuit défectueux est généré pour chaque composant du type spécifié dans le circuit. Les transistors et les diodes sont supportés.

```
&montecarlo <nombre_échantillons> <écart_type>
```

Une analyse Monte Carlo est effectuée selon le nombre spécifié d'échantillons. Les résistances, capacités, largeur et longueur de transistors MOS et facteurs de surface des BJT sont modifiés aléatoirement selon une distribution gaussienne et la valeur d'écart type fournie.

```
&sizing <type_composant> <paramètre> <nombre_candidats>
        <déviation_relative>
```

Une liste de circuits candidats de dimensionnement est générée et simulée. Pour chaque candidat, la valeur du paramètre spécifié est variée de plus ou moins la déviation relative donnée et ce, pour chaque instance du type de composant spécifié.

ANNEXE C
Applications développées

Cette annexe décrit de façon plus détaillée les applications de simulation de pannes, de dimensionnement automatique de circuits et d'analyse Monte-Carlo qui ont été développées dans le cadre de ce mémoire. Nous y aborderons tout d'abord le fonctionnement général de ces applications, puis un guide d'utilisation sera présenté.

C.1 Fonctionnement général

Les applications mentionnées dans ce mémoire ont été développées à l'intérieur d'un seul et même logiciel. Celui-ci a été programmé en langage C++ et peut être compilé sur n'importe quel système Unix. Puisqu'il fait appel à certaines commandes POSIX, notamment en ce qui touche les communications *socket*, il ne peut être compilé sous Windows sans quelques modifications. Une interface graphique est incluse dans le logiciel afin de permettre à l'utilisateur de définir rapidement et facilement les circuits à simuler ainsi que de visionner les résultats de simulation. La librairie graphique QT 3.1 a été utilisée pour générer toutes les fenêtres de l'interface graphique utilisateur. Cette librairie est compatible avec les systèmes Unix ou Windows. Les fonctionnalités offertes par le logiciel peuvent être regroupées en quatre catégories :

- Communes à toutes les applications,
- Dédiées à la simulation de pannes,
- Dédiées au dimensionnement automatique de circuits,
- Dédiées à l'analyse Monte Carlo.

La Figure C.1 illustre la répartition des diverses fonctionnalités entre ces catégories. De façon générale, les fonctionnalités reliées aux applications permettent à l'utilisateur de définir de quelle manière sont générés les circuits modifiés à simuler. De même, elles permettent de convertir les résultats bruts de simulation en résultats d'intérêt reliés à l'application, de façon à éviter à l'utilisateur d'effectuer des calculs manuellement.

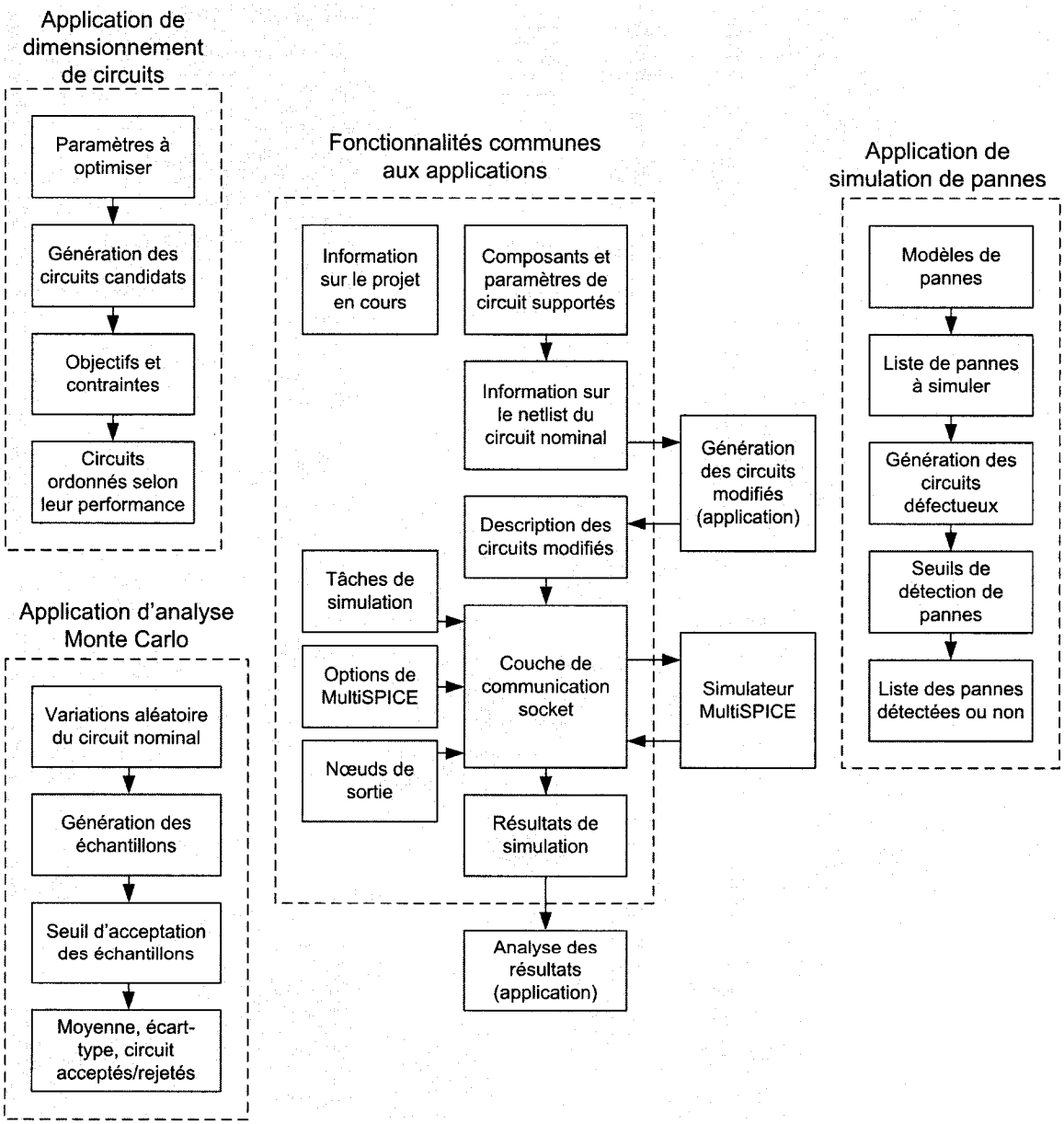


Figure C.1 : Schéma interne des applications de simulations multiples

C.2 Guide d'utilisation

Cette section présente les principaux éléments nécessaires pour faire fonctionner les applications de simulation de pannes, de dimensionnement de circuits et d'analyse Monte Carlo.

C.2.1 Entrées

Lors de la création d'un nouveau projet, les fichiers suivants, qui contiennent les options par défaut, doivent être présents dans le même répertoire que l'exécutable du logiciel :

`analyses.dat` : Définition des tâches de simulation disponibles (DC / AC / transitoire)
`models.dat` : Types de composants de circuit supportés (dépend du simulateur utilisé)
`options.dat` : Options de simulation disponibles pour toutes les applications
`faultoptions.dat` : Options spécifiques à la simulation de pannes
`sizingoptions.dat` : Options spécifiques au dimensionnement de circuits
`mcoptions.dat` : Options spécifiques à l'analyse Monte Carlo

Lorsque le projet est créé, une copie de ces fichiers est enregistrée dans le répertoire du projet, permettant de définir des options propres à ce projet. Tous les noms de fichiers contenant les informations sur le projet sont du type `<netlist>.<type_fichier>`. Les fichiers suivants se retrouvent donc dans le répertoire de chaque projet :

`<netlist>.job` : Copie de `analyses.dat` avec les tâches propres au projet
`<netlist>.mod` : Copie de `models.dat` avec les composants propres au projet
`<netlist>.nom` : Information provenant du *netlist* du circuit nominal
`<netlist>.opt` : Copie de `options.dat` avec les options propres au projet
`<netlist>.pro` : Information sur le type de projet et localisation des fichiers d'intérêt
`<netlist>.fls` : Liste de pannes à injecter pour la simulation de pannes

`<netlist>.fso` : Copie de `faultoptions.dat` avec les options propres au projet

`<netlist>.dev` : Déviations à appliquer par l'analyse Monte Carlo sur les composants

`<netlist>.mco` : Copie de `mcoptions.dat` avec les options propres au projet

`<netlist>.del` : Paramètres à optimiser pour le dimensionnement de circuits

`<netlist>.szo` : Copie de `sizingoptions.dat` avec les options propres au projet

Lors du chargement d'un projet existant, toute l'information de la session précédente est récupérée, à l'exception des résultats de simulation.

C.2.2 Sorties

Les fichiers suivants apparaissent dans le répertoire du projet et de l'information s'y ajoute au fur et à mesure que des opérations sont effectuées :

`iterations.trace` : Fichier de trace pour toutes les opérations effectuées par le simulateur *MultiSPICE*. On y retrouve de l'information sur la convergence de chacun des circuits.

`comm.trace` : Fichier qui contient toutes les transactions effectuées via la communication *socket* entre l'application et le simulateur *MultiSPICE*.

C.2.3 Menus

Les menus suivants sont disponibles pour l'utilisateur :

Project : Création d'un nouveau projet ou ouverture d'un projet existant, sauvegarde et fermeture du projet en cours, démarrage et arrêt du simulateur, sortie de l'application.

Netlist : Affichage des composants de circuit et paramètres supportés, affichage des composants, valeurs des paramètres et nœuds du fichier *netlist* actuellement ouvert, sélection des nœuds de sortie et des composants / paramètres d'intérêt.

Simulation : Modification des options de simulation, définition des tâches de simulation à effectuer, démarrage des simulations et sauvegarde des résultats de simulation.

Fault simulation : Définition de la liste de pannes à simuler, modification des options de simulation de pannes, définition de l'analyse Monte Carlo si requis, génération et affichage des circuits défectueux, affichage des résultats de simulation pertinents à la simulation de pannes.

Automatic sizing : Spécification des paramètres à optimiser, génération et affichage des circuits candidats, affichage des résultats de simulation pertinents au dimensionnement automatique de circuits, sauvegarde de la liste ordonnée des circuits candidats.

Monte Carlo : Spécification des variations de paramètres à appliquer, modification des options d'analyse Monte Carlo, génération et affichage des circuits échantillons, affichage des résultats de simulation pertinents à l'analyse Monte Carlo.