Modélisation comportementale des circuits analogiques et mixtes

F. Lemery

Thèse

présentée par

François LÉMERY

pour obtenir

LE TITRE DE DOCTEUR DE

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 mars 1992)

(Spécialité: **Microélectronique**)

==========

MODÉLISATION COMPORTEMENTALE DES CIRCUITS ANALOGIQUES ET MIXTES

==========

Date de soutenance: **20 décembre 1995**

Composition de jury:

Président:	Bernard COURTOIS
Rapporteurs:	Daniel AUVERGNE
	Michel LE HELLEY
Examinateurs:	Meryem MARZOUKI
	Joseph BOREL
	Jean-Paul MORIN

Thèse préparée à

SGS-Thomson Microelectronics, R&D Centrale, Centre de Crolles

Contrat CIFRE SGS-Thomson et TIMA

“Le langage est aux postes de commande de l’imagination.”
G. Bachelard.

À Mireille,

Résumé

Pour pouvoir intégrer sur une seule puce des systèmes toujours plus complexes comportant à la fois des fonctions numériques et analogiques, l'utilisation d'une méthodologie de conception hiérarchique est indispensable. Basée sur la modélisation comportementale de chaque élément du circuit, avant tout choix d'architecture, une telle approche permet en effet de réduire les temps de simulation, de conception et d'améliorer la fiabilité. Appliqué avec succès dans le domaine digital, ce paradigme doit maintenant être étendu à l'analogique. Cela est aujourd'hui possible grâce à l'offre récente de puissants langages de modélisation comportementale analogique et mixte.

Cette thèse a permis d'introduire l'utilisation de ces langages au sein de la communauté des concepteurs, par le développement d'un environnement CAO d'aide à la conception de modèles analogiques et mixtes. Il est basé sur une *bibliothèque fonctionnelle* adaptée à la modélisation de circuits élémentaires (amplificateurs opérationnels) mais aussi de systèmes très complexes, tels qu'un système de sécurité *air-bag*. Plusieurs techniques de description ont été abordées: macro-modélisation SPICE et modélisation comportementale à l'aide de plusieurs langages dont les propriétés ont été comparées (FAS, CFAS, HDL-A et MAST). Cet environnement comporte aussi un *outil de caractérisation analogique* qui permet de générer rapidement les paramètres des modèles en fonction de mesures des performances du circuit associé, par des simulations électriques. En outre, pour faciliter les échanges de modèles et transférer des bibliothèques vers des langages différents, des *traducteurs automatiques* ont dû être élaborés, tels que FAS vers CFAS, FAS vers MAST et FAS vers HDL-A.

Abstract

To integrate on one single chip ever more complex systems composed of both digital and analog functions, a hierarchical design methodology is needed. Based on a behavioural modelling of each circuit element, before any architectural choice, such an approach reduces simulation and design times and increases reliability. Successfully applied in the digital domain, this paradigm should now be extended to the analog one. This is now possible due to the recent availability of powerful languages for analog and mixed behavioural modelling.

This thesis introduces the usage of such languages in the designer community through the development of a CAD environment providing help for analog and mixed model design. It is based on a *functional library* dedicated to the modelling of basic circuits (e.g. operational amplifiers) but also of very complex systems such as a security *air-bag* system. Different description techniques have been studied: SPICE macro-modelling and behavioural modelling with several languages, whose properties have been compared (FAS, C-FAS, HDL-A and MAST). This environment is also composed of an *analog characterization tool* which is used to rapidly compute model parameters from performance measurements of the associated circuit, through electrical simulations. Moreover, to ease model exchange and transfer libraries into different languages, *automatic translators* have been developed such as FAS to CFAS, FAS to MAST and FAS to HDL-A.

Avant-propos et Remerciements

Cette thèse a été effectuée dans le cadre d'une convention CIFRE entre le laboratoire TIMA de l'INPG et la R&D Centrale de SGS-Thomson Microelectronics du site de Crolles, et plus spécialement le service "Design Solutions" qui propose des plateformes CAO adaptées aux technologies de la compagnie.

Je souhaiterais tout d'abord remercier Monsieur Joseph Borel, responsable de ce service, pour m'avoir permis de travailler sur ce sujet extrèmement prometteur et relativement nouveau en ce qui concerne les circuits analogiques. Je tiens d'autre part à lui exprimer ma profonde gratitude pour avoir accepté de faire partie du jury.

Je remercie tout aussi chaleureusement Monsieur Bernard Courtois, directeur du laboratoire TIMA et co-directeur de thèse, qui a permis que cette collaboration démarre et réussisse. Qu'il soit de plus assuré de ma reconnaissance pour avoir accepté d'être Président du jury.

Tous mes remerciements vont à Messieurs Daniel Auvergne et Michel Le Helley pour leur lecture attentive du manuscrit en tant que rapporteurs de cette thèse.

Enfin, je voudrais exprimer mes remerciements les plus sincères à Madame Meryem Marzouki, co-directrice de thèse, et Monsieur Jean-Paul Morin, responsable industriel de ces travaux, pour leur efficacité dans l'encadrement de ce projet. Leurs conseils ont été essentiels pour choisir les axes prioritiares d'étude et pour mener ce projet à terme.

Cette thèse n'aurait pas été aussi enrichissante sans la présence et l'expérience de nombreuses personnes. Je pense en particulier à Monsieur Eric Nercessian, collègue de travail, qui a fortement contribué au développement de l'environnement de caractérisation. De même, je souhaiterais exprimer toute ma gratitude à Monsieur Pierre Saramito, actuellement chercheur au CNRS, qui est le créateur d'un environnement de conception de traducteurs et qui a ouvert la voie à des développements présentés dans ce document. Enfin, je tiens à saluer l'effort des ingénieurs du support des logiciels de simulation de la société Anacad, largement utilisés au cours de ces travaux, et en particulier de Messieurs Daniel Borgraeve et Olivier Schnitzler.

Je voudrais d’autre part signaler que ce projet s’est inscrit dans le cadre d’une participation au projet européen JESSI (*Joint European Submicron Silicon*) et a permis en particulier de collaborer avec les sociétés Anacad et BOSCH.

Enfin, je tiens à exprimer ma profonde sympathie à tous ceux qui ont contribué de près ou de loin au bon déroulement de ces trois ans, et tout particulièrement Monsieur Nazir Hakim pour son assistance logistique extrèmement précieuse et Monsieur Davoud Samani pour son accueil chaleureux dans la société et son engagement dans la définition initiale du sujet de recherche.

Bien sûr, je ne saurais oublier mes parents, ma soeur et enfin mon épouse, qui m’ont prodigué des encouragements constants.

Table des Matières

Liste des Figures

Liste des Tableaux

Chapitre 1: Introduction

1 Circuits mixtes analogiques/digitaux

Du fait des développements technologiques de ces dernières années, sont aujourd'hui intégrés sur une seule puce des systèmes électroniques qui étaient jusqu'à présent réalisés sous forme de cartes. Cette tendance à l'intégration et à la miniaturisation des circuits est portée par le développement "explosif" des applications multi-media, de télécommunications et automobiles. De tels systèmes comportent un nombre toujours croissant de modules pouvant appartenir à des domaines différents: des fonctions *numériques,* prédominantes, qui sont basées sur des micro-processeurs ou micro-contrôleurs, des mémoires et des blocs DSP de traitement du signal (Digital Signal Processing), mais aussi des fonctions *analogiques* d'amplification et de filtrage qui se trouvent en particulier dans les circuits de conversion *analogique/digitale* (A/D) en entrée et *digitale/analogique* (D/A) en sortie. Certaines technologies (Bipolar-CMOS-DMOS) permettent même d'intégrer des fonctions analogiques de puissance avec de la logique.

Un exemple typique de ces nouvelles applications est le système de sécurité automobile nommé *air-bag*: à la fonction de base correspondant au traitement analogique du signal d'accélération pour déclencher l'électro-valve d'air comprimé, se rajoutent de nombreux éléments de contrôle numérique, qui testent en permanence le bon fonctionnement du système.

Depuis dix ans, la conception des fonctions digitales a été fortement automatisée par le développement d'outils de conception assistée par ordinateur (CAO) très avancés. Citons par exemple l'apparition de logiciels de synthèse qui permettent de générer un circuit constitué de portes logiques à partir d'une simple description de sa fonction. Or, la conception de circuits mixtes où les blocs analogiques et digitaux ne peuvent être dissociés pour une étude précise du fonctionnement, reste particulièrement difficile, essentiellement du fait d'un manque d'outils semblables du côté analogique. Une transposition vers l'analogique des outils et

méthodologies qui ont fait le succès du digital doit donc être effectuée. En particulier, un des points clés à résoudre concerne la modélisation comportementale des circuits analogiques et mixtes.

2 La modélisation comportementale

2.1 Définition

La validation d'un système électronique, qu'il soit intégré sur une puce ou réalisé sous forme de cartes, est encore essentiellement basée sur des logiciels de *simulation*[1]. Ceux-ci font appel à des modèles des différents éléments utilisés, qui en décrivent le *comportement*, c'est à dire les relations entre les signaux présents sur les points d'entrée/sortie (E/S).

Ainsi, lors de la conception de circuits intégrés tels qu'un amplificateur opérationnel ou une porte logique, des *modèles de composants* (transistors, diodes, résistances, capacités,...) sont utilisés par un simulateur électrique, dit analogique (le programme *SPICE es*t sans doute le plus connu). Ces modèles décrivent les relations macroscopiques entre tensions et courants des diverses bornes, sous forme d'équations différentielles. Il s'agit donc d'une représentation mathématique de phénomènes physiques auxquels obéissent les composants.

La modélisation comportementale désigne plutôt une représentation fonctionnelle de haut-niveau, par opposition à une représentation structurelle, et qui est indispensable à la validation de circuits complexes, comportant un grand nombre d'amplificateurs ou de portes. En effet, les capacités de convergence des simulateurs analogiques sont fortement conditionnées par le nombre de transistors. Même si de nouvelles techniques de simulation plus rapides sont développées, cette limitation ne peut être résolue qu'en adoptant une approche hiérarchique multi-niveaux, consistant à décomposer le système en un ensemble de blocs fonctionnels. Le schéma de chaque bloc, ou de seulement certains d'entre eux, peut alors être remplacé par une description approchée uniquement fonctionnelle et plus abstraite. La définition de cette représentation est l'objet de la *modélisation comportementale*.

1. Les études dans le domaine de la preuve formelle sont en effet encore trop récentes pour proposer des outils de vérification formelle utilisables en pratique.

2.2 Exemples analogiques et digitaux

Les modules numériques sont ainsi décrits par des équations booléennes, des tables de vérité ou des tables d'états en précisant les temps de propagation entre E/S. Les signaux manipulés ne sont plus électriques mais abstraits: des bits définis par des états logiques (0/1/ indéterminé) ou des mots de bits ou même des fichiers de données peuvent être transmis entre modèles. Un autre type de simulateur, dit *digital*, est ici utilisé: son fonctionnement est souvent dirigé par les évènements que constituent les changements d'état des signaux.

Dans le domaine analogique, les signaux restent des grandeurs électriques, fonctions continues du temps (ou de la fréquence). Les modèles comportementaux analogiques peuvent être représentés par un ensemble simplifié d'équations différentielles, des fonctions mathématiques non-linéaires ou linéaires par morceaux ou des tables de données. Notons que le terme de *macro-modélisation* est couramment employé en analogique. Il désigne en fait une méthode particulière de modélisation comportementale qui consiste en une simplification du schéma et, essentiellement, en l'utilisation de sources idéales contrôlées, proposées par les simulateurs de type SPICE, pour exprimer des relations entre tensions et courants. Cette approche est cependant assez limitée aux primitives du simulateur, introduit des effets parasites et peut poser des difficultés de convergence.

2.3 Méthodologies de conception

La modélisation comportementale est indispensable pour la validation de systèmes complexes, afin de résoudre les problèmes de convergence des simulateurs. Mais elle permet surtout de mettre en oeuvre la *méthodologie de conception hiérarchique descendante*, indispensable pour réaliser un circuit répondant du premier coup aux spécifications.

Afin de permettre au concepteur de choisir, à chaque niveau hiérarchique, l'architecture adaptée à ses besoins, chaque cellule susceptible d'être utilisée doit être au préalable associée à un modèle comportemental. La simulation des modèles comportementaux permet alors de valider l'architecture et de fixer des contraintes sur les caractéristiques des divers blocs fonctionnels. La structure de chaque bloc est alors étudiée selon le même procédé, jusqu'au niveau du schéma de transistors.

Enfin, après réalisation du layout et extraction des parasites (résistances, capacités) de chaque bloc élémentaire, un modèle représentatif de ses performances et défauts doit être

généré. Il est alors utilisé, selon une *approche ascendante*, pour une étude précise mais rapide des propriétés du circuit conçu, en fonction des contraintes technologiques. À la fin, un modèle comportemental réaliste du circuit est obtenu. Il peut être utilisé pour une étude du système à plus haut niveau.

2.4 Difficultés liées au domaine analogique/digital

2.4.1 Absence de langage standard

Une des clés du succès pour l'application d'une telle méthodologie est l'existence d'un langage de description qui soit largement accepté comme standard. C'est le cas dans le domaine digital où des langages standards tels que *Verilog et VHDL* ont joué le rôle de catalyseur pour le développement de bibliothèques de cellules et de leurs modèles associés.

L'application de cette méthodologie aux circuits mixtes analogiques-digitaux (convertisseurs A/D et D/A, boucle à verrouillage de phase ou PLL) est assez récente et est liée à l'avènement de nouveaux simulateurs mixtes offrant des possibilités de modélisation comportementale. Cependant, un langage mixte, qui soit compatible pour la description digitale avec un des standards digitaux, n'est pas encore disponible. Dans ce but, un comité de standardisation (*IEEE 1076.1*) tente de définir les extensions analogiques de *VHDL* pour l'élaboration d'un sur-ensemble nommé *VHDL-A*. Un tel langage permettra réellement de résoudre les problèmes de conception d'applications mixtes et même multi-disciplinaires.

2.4.2 Contenu des modèles analogiques

Une autre difficulté est le manque de consensus sur le contenu des modèles analogiques. En effet, de nombreux phénomènes peuvent être pris en compte en plus de la fonction nominale: les éléments parasites d'entrée/sortie, les non-linéarités, le bruit, la dépendance des paramètres par rapport à la température, leurs variations en fonction des fluctuations statistiques technologiques... De plus, dans le cas des circuits échantillonnés et à découpage, les représentations temporelles et fréquentielles sont très différentes. En fait, le contenu des modèles analogiques dépend surtout de la précision souhaitée et des analyses envisagées par l'utilisateur.

C'est pourquoi, pour faciliter l'écriture des équations de ces modèles, des outils doivent être étudiés et des méthodologies définies. Cela peut concerner soit la génération des équations

à partir du schéma du circuit, soit la simple détermination des paramètres.

3 Les solutions proposées

Au terme de cette introduction sur les avantages de la modélisation comportementale et l'état de l'art dans le domaine des circuits analogiques et mixtes, il apparaît clairement qu'une méthodologie et un environnement de conception de modèles analogiques et mixtes doivent être définis pour faciliter et automatiser le travail de modélisation des concepteurs. C'est une des contributions de cette thèse qui a donné lieu au développement de trois catégories d'outils:

- *une bibliothèque de modèles fonctionnels*: il s'agit d'une bibliothèque fonctionnelle comportant des modèles analogiques et analogiques/digitaux, paramétrables, génériques, c'est-à-dire indépendants de la structure et de la technologie, et prenant en compte un certain nombre de non idéalités (non-linéarités, saturations, constantes de temps...). Le but essentiel de cette bibliothèque est de fournir des solutions quant à la manipulation des différents types de signaux rencontrés: analogiques, digitaux, quantifiés et échantillonnés. Deux niveaux de modèle sont considérés: les modèles de haut-niveau (fonctions mathématiques, amplificateurs, comparateurs, échantillonneurs, convertisseurs, modulateurs, bascules...) qui sont utilisés pour des études de systèmes, et des modèles de plus bas-niveau qui sont adaptés à la construction de modèles comportementaux d'amplificateurs opérationnels (étages d'entrée, de gain, de sortie,...). Cette bibliothèque a été par exemple utilisée pour l'étude d'un système de déclenchement d'*air-bag* et la modélisation d'un nouvel amplificateur de SGS-Thomson.
- *un outil de caractérisation analogique*, dont le but est de générer rapidement par des simulations électriques les paramètres d'un modèle en fonction des performances du circuit qu'il doit représenter. Alors que dans le domaine digital, il est seulement nécessaire de déterminer des caractéristiques temporelles, la caractérisation des performances des circuits analogiques et mixtes est bien plus complexe car elle fait intervenir de nombreuses mesures déterminées soit par l'analyse du point de fonctionnement, soit par l'analyse transitoire ou fréquentielle. Cet outil permet au concepteur de définir une bibliothèque de schémas de caractérisation ou procédures de mesure, dont l'exécution est automatiquement gérée afin de calculer les paramètres des modèles. Un langage de spécification des mesures a de plus été défini pour piloter la caractérisation.

- *des traducteurs entre différents langages de modélisation (Fas, C-Fas, HDL-A, Mast)*: en l'absence de standard et pour faciliter l'échange de modèles entre des sociétés n'utilisant pas les mêmes langages de description, il s'est en effet avéré nécessaire d'élaborer une méthodologie de traduction automatique. D'autre part, une telle traduction est également intéressante pour transférer rapidement des bibliothèques de modèles d'un langage à l'autre. Plusieurs traducteurs ont été ainsi développés (*Fas* vers *C-Fas*, *Fas* vers *HDL-A*, *Fas* vers *Mast*) et ont été utilisés avec succès.

4 Organisation du mémoire

Ce mémoire est divisée en deux parties. La première, constituée des chapitres 2 à 3, détaille les différents objectifs et avantages de la modélisation comportementale et l'état de l'art pour les circuits analogiques et mixtes. En particulier, le chapitre 2 décrit les techniques de simulation analogique et mixte, les méthodologies hiérarchiques de conception descendante et ascendante et enfin introduit l'utilisation de la modélisation comportementale pour la simulation et le diagnostic de fautes. Le chapitre 3 expose, quant à lui, les méthodologies de modélisation comportementale analogique et mixte, c'est à dire les différents types de langage, soit structurel soit comportemental, les méthodes de définition des équations et les outils de calcul des paramètres.

La seconde partie, constituée des chapitres 4 à 6, traite de l'*environnement de conception de modèles comportementaux analogiques et mixte*, qui a fait l'objet de ces travaux. Le chapitre 4 traite de la bibliothèque de modèles fonctionnels et présente les différentes méthodes de modélisation mises en oeuvre. Le chapitre 5 est relatif à l'environnement de caractérisation automatique de composants analogiques. Enfin, le chapitre 6 est consacré à la présentation des méthodologies utilisées pour la traduction automatique de modèles d'un langage à l'autre.

Les annexes 1 à 8, fournies avec le document, décrivent les détails d'implémentation des différents éléments de l'environnement réalisé et présentent les principales publications obtenues.

Chapitre 2: Les applications des modèles comportementaux

1 Introduction

L'utilité de la modélisation comportementale pour la conception des circuits digitaux VLSI ne fait plus aucun doute. Elle permet en effet de *réduire les temps de conception* et de concevoir des circuits de plus grande *qualité* pour deux raisons essentielles:

- la simulation comportementale d'un circuit complexe est beaucoup plus rapide qu'une simulation effectuée avec une description "transistors": le concepteur peut donc mieux vérifier le fonctionnement du circuit,
- la description comportementale de chaque bloc du circuit conduit à une définition très précise de ses spécifications, ce qui permet d'éviter des erreurs de conception et d'obtenir un circuit optimal.

Ce chapitre tente de mettre en valeur les avantages qu'apporte la modélisation comportementale tant au niveau de la simulation analogique et mixte, qu'au niveau des méthodologies de conception hiérarchique, et même au niveau de la simulation et du diagnostic de fautes.

2 Validation par la simulation

La simulation est essentielle dans la conception des circuits intégrés VLSI en tant qu'outil de validation des choix du concepteur. Pour des raisons de compétitivité, elle doit être la plus rapide et la plus fiable possible. On distingue généralement trois types de simulation:

- la ***simulation analogique ou électrique***: elle traite des signaux continus dans le temps et est utilisée pour déterminer les performances électriques des circuits. Le niveau de précision

demandé est généralement élevé et de ce fait, le temps de simulation des circuits comportant un grand nombre de transistors est souvent critique.

- la ***simulation digitale***: couramment utilisée pour les circuits digitaux VLSI, elle manipule des signaux discrétisés et quantifiés (0,1, indéterminé ou X,...) et se caractérise par une très grande rapidité.
- la simulation ***mixte analogique-digitale***: elle permet d'accélérer l'étude temporelle des circuits mixtes complexes.

Les caractéristiques des simulateurs électriques et mixtes sont tout d'abord présentés (on se reportera à l'Annexe 1 pour plus de détails sur les algorithmes employés). Sont ensuite cités quelques simulateurs spécifiques.

2.1 La simulation électrique

La référence en matière de simulateur analogique de circuits intégrés est le programme SPICE [Nag75], développé à l'université de Berkeley. Il a donné lieu à de nombreuses versions industrielles basées sur un même langage de description structurelle, nommé dans ce document langage SPICE. Une bibliothèque de composants modélisés dans le code même du simulateur est fournie et comporte des éléments passifs (résistances, capacités, inductances, inductances mutuelles), des composants semi-conducteurs (diodes, transistors bipolaires, à effet de champ JFET et MOSFET), des sources idéales indépendantes de tension et de courant et enfin des sources idéales contrôlées polynômiales (sources de tension ou courant contrôlées par des tensions ou des courants). L'écriture de nouveaux modèles de composants est une tâche difficile de programmation, qui dépend des algorithmes utilisés par le simulateur. Elle est donc réservée à des spécialistes.

Des simulateurs plus récents, tels que Saber [Sab87], Eldo [Eld94], Spectre [Spe95] proposent, quant à eux, des langages de description comportementale, qui facilitent l'écriture de nouveaux modèles. Ils possèdent d'autre part une bibliothèque étendue de primitives comportant des modèles comportementaux de macro-blocs (amplificateurs, comparateurs, commutateurs analogiques, portes logiques, etc...).

La première étape effectuée par ces programmes consiste en la ***mise en équation*** du réseau électrique par application des lois de Kirchhoff. Signalons que la taille du système

d'équations est une fonction exponentielle du nombre de noeuds et conditionne donc fortement la vitesse de simulation.

Plusieurs types d'analyse peuvent alors être réalisés pour étudier le comportement du circuit:

- l'étude du ***point de fonctionnement*** du circuit ou analyse DC qui correspond à une étude en régime permanent (cf Annexe 1),
- l'étude de la ***réponse temporelle*** dite analyse ***transitoire*** (cf Annexe 1),
- l'étude de la ***réponse fréquentielle*** ou petits signaux (analyse AC), pour laquelle le circuit est linéarisé autour du point de fonctionnement (cf Annexe 1),
- les analyses de ***bruit***, généralement fréquentielles, mais il existe aussi des techniques de simulation transitoire de bruit [Bol94], utilisées par exemple dans le simulateur Eldo.
- l'étude de la ***sensibilité***, qui consiste en la définition du pourcentage de variation de grandeurs électriques du circuit en fonction de certains paramètres de conception, en linéarisant le circuit autour d'un point de polarisation.
- la définition des ***pôles*** et ***zéros***, à la suite d'une analyse fréquentielle, par exemple par l'algorithme QZ de recherche de valeurs propres [Eld94],
- les analyses ***statistiques*** de type *Monte-Carlo* afin de déterminer la dispersion des performances du circuit en fonction des fluctuations statistiques de paramètres de conception. Un grand nombre de simulations sont ici requises. Cette étude permet ensuite de définir la valeur nominale des composants pour obtenir un rendement optimal.

Ces simulateurs doivent faire face à des problèmes de convergence qui apparaissent essentiellement lors de la recherche du point de fonctionnement pour les circuits comportant un nombre élevé de transistors, ainsi que pour les circuits fortement couplés, en particulier en présence de transistors bipolaires. D'autre part, certaines applications à échantillonnage, telles que les filtres à capacités commutées, les boucles à verrouillage de phase (PLL) ou encore les alimentations à découpage, requièrent des temps de simulation temporelle relativement élevés. En effet, alors que la fréquence des horloges d'échantillonnage est de l'ordre du MHz, celle du signal qui est traité par le système est plutôt de l'ordre du kHz. De très nombreuses périodes d'horloge sont donc nécessaires pour observer les phénomènes souhaités, tels que le temps de capture d'une PLL. Enfin, concernant les circuits analogiques-digitaux, des simulateurs

digitaux particulièrement rapides doivent être bien-sûr mis en oeuvre pour la partie numérique.

2.2 La simulation analogique-digitale

La simulation mixte analogique-digitale permet d'étudier le comportement temporel de systèmes complexes en un temps extrêmement réduit par rapport à une simulation uniquement électrique. Ce type de simulation est en effet basé sur l'abstraction de la partie digitale à un niveau fonctionnel logique. Pour cette partie, les grandeurs étudiées ne sont donc plus électriques mais numériques et sont caractérisées par leurs changements d'état. Des algorithmes dirigés par évènements dits *event-driven* permettent d'étudier de manière très efficace l'évolution des signaux digitaux.

Une simulation mixte peut être décomposée en trois phases distinctes:

- une phase d'***élaboration***: elle correspond au partitionnement du circuit mixte en blocs distincts analogiques et digitaux, chaque partie étant traitée par les algorithmes concernés. Aux interfaces entre les deux parties, doivent être placés des modèles plus ou moins élaborés de ***convertisseurs*** A/D et D/A, qui assurent la correspondance des données entre les algorithmes analogiques et digitaux (cf Annexe 1).
- une phase d'***initialisation***: il s'agit de déterminer le point de fonctionnement du système, c'est à dire l'état initial de toutes les grandeurs mises en jeu (tensions, courants, états logiques). Cette recherche est indispensable au simulateur analogique et correspond à une analyse DC. Pour la partie digitale, cette notion dépend du simulateur: cela peut correspondre soit à une initialisation (solution au temps 0), soit à un certaine durée, appelée temps de *setup*, au bout de laquelle un état stable est trouvé.
- une phase de ***simulation***: elle doit résoudre les problèmes de ***synchronisation*** des algorithmes électriques et digitaux, qui ont des gestions différentes du pas de temps (cf Annexe 1).

Au cours de ce travail, le simulateur mixte Verilog-Eldo a par exemple été utilisé lors de l'étude d'un système analogique-digital de déclenchement d'air-bag, décrit en Annexe 3. Il est basé sur deux programmes distincts Eldo et Verilog [Ver95] qui communiquent par l'intermédiaire d'un logiciel nommé *backplane*. Notons que le simulateur Eldo possède lui-même un algorithme de simulation digitale qui est en particulier exécuté lors de l'utilisation du langage de modélisation mixte HDL-A [Hdl94]. Bien qu'il soit beaucoup moins puissant

qu'un réel simulateur digital, il a tout de même été utilisé avec succès dans le cas de la modélisation HDL-A de l'air-bag (cf Chapitre 4). Il en est de même du simulateur Saber qui est associé au langage de description mixte Mast [Mas90].

2.3 Simulateurs spécifiques

Pour accélérer l'étude de certains types de circuits intégrés, des algorithmes spécialisés ont été développés: les uns sont dédiés aux *filtres à capacités commutées*, comme Switcap [Fan83], d'autres concernent l'approximation des réponses temporelles de circuits *linéarisés*, comme la technique ***AWE*** (*Asymptotic Waveform Evaluation*) [PiR90] [RRP93]. Cette dernière consiste à déterminer les racines dominantes (pôles ou zéros) en réduisant l'ordre du système. Précisons que cette méthode *AWE* a été développée pour déterminer les délais de propagation des circuits digitaux en construisant des modèles d'interconnexion plus précis que les traditionnels modèles RC.

Enfin, des simulateurs ***symboliques***, tels que ISAAC [GWS89] ou ASAP [FRH91], sont utilisés essentiellement dans le cadre de la synthèse analogique pour l'optimisation de paramètres de conception. Applicable à des circuits *linéarisés*, l'analyse symbolique consiste à déterminer des *équations analytiques* de certaines caractéristiques du circuit (fonctions de transfert, impédances d'E/S, facteurs de réjection d'alimentation) en fonction des éléments du circuit ou symboles. Ces formules peuvent être ensuite évaluées très rapidement. D'autres applications existent comme l'analyse de tolérance, des sensibilités, des pôles et zéros ou même pour le diagnostic de fautes [MaP93]. Cependant, le nombre de termes augmente exponentiellement en fonction du nombre de noeuds du réseau. Des simplifications doivent être effectuées ce qui réduit le domaine de validité des formules [DoD92] [FRM92] [WFG94].

3 Méthodologies de conception hiérarchique

Quelles que soient les méthodes utilisées en simulation analogique, électrique ou symbolique, le nombre de composants pouvant être traités sera toujours limité. Il est donc nécessaire de changer de paradigme et d'adopter une méthodologie de conception hiérarchique descendante dite "*top-down*", basée sur la modélisation comportementale, et qui décompose le

problème de la conception d'un système complexe en une suite de problèmes élémentaires, plus faciles à appréhender.

Cette méthodologie hiérarchique permet d'autre part d'assurer une conception de qualité en évitant tous les problèmes de sur-dimensionnement des éléments du circuit, qui sont inhérents à l'approche traditionnelle ascendante dite "*bottom-up*" [GDW92]. Celle-ci consiste à élaborer en premier lieu les blocs constitutifs du circuit au niveau transistor, leurs spécifications ne pouvant être précisément définies que lorsqu'ils sont assemblés en la fonction souhaitée. Outre les problèmes de simulation qui sont posés pour valider le fonctionnement du circuit complet, un grand nombre d'itérations de conception sont inévitablement requises. Pour les limiter, un sur-dimensionnement des spécifications peut être effectué, conduisant en un circuit peu optimal.

C'est pourquoi la méthodologie *top-down* est depuis longtemps appliquée à la conception et synthèse des circuits digitaux. Le but visé est d'obtenir au premier coup un circuit répondant aux spécifications [GDW92]. Les simulations mettant en jeu les modèles comportementaux permettent en effet d'éliminer les erreurs de choix architecturaux dès les premières phases de la conception. En raison de l'intégration croissante de fonctions analogiques, telles que dans les convertisseurs A/D et D/A [LSG91] ou les boucles à verrouillage de phase (PLL) [MLN93], elle est aujourd'hui appliquée dans le domaine analogique. *Liu et al.* [LSG91] désignent cette stratégie: ***conception top-down dirigée par les contraintes*** car la vérification a lieu au plus tôt et procède par propagation descendante des spécifications ou contraintes.

Après avoir précisé les différents niveaux hiérarchiques qui sont définis dans les domaines digitaux et analogiques, nous décrirons les deux phases principales de conception: la première phase *descendante de décomposition* du système et la seconde phase *ascendante de validation*, après réalisation de l'architecture de chaque bloc fonctionnel et surtout après élaboration du *layout*.

3.1 Les niveaux d'abstraction

3.1.1 Domaine digital

Les niveaux de description, définis dans le domaine digital par *Gajski et al.* [GDW92], sont décrits par le Tableau 1: système, micro-architecture, logique et circuit. Chaque niveau est caractérisé par deux types de représentations qui sont successivement utilisées dans la

méthodologie top-down: la représentation *comportementale*, indépendante de toute architecture, et la représentation *structurelle*, qui décrit une architecture donnée à l'aide d'éléments appartenant au niveau inférieur.

Pour décrire le comportement, des fonctions de transfert et des diagrammes temporels sont utilisés au niveau "circuit", des équations booléennes et des diagrammes d'états au niveau "logique". La description "registre de transfert" au niveau "micro-architecture" spécifie pour chaque état de contrôle la condition à tester, les transferts de registre à exécuter et l'état suivant. Enfin, des schémas synoptiques et des langages algorithmiques sont employés pour définir la fonction du "système".

Tableau 1 Niveaux d'abstraction des systèmes digitaux [GDW92]

Niveau d'abstraction	Représentation comportementale	Représentation structurelle
Système	Schémas synoptiques, Algorithmes	Processeurs, Mémoires
Micro-architecture	Registre de transfert (RTL)	Registres, ALUs
Logique	Équations booléennes, Diagrammes d'états	Portes logiques
Circuit	Fonctions de transfert, Diagrammes temporels	*Transistors interconnectés*

La *synthèse* correspond au passage de la représentation comportementale à la représentation structurelle. Des outils de synthèse automatique entre la représentation comportementale "registre de transfert" et la description structurelle en "portes logiques" sont par ailleurs couramment utilisés dans l'industrie [DCT94]. D'autres outils de synthèse de plus haut-niveau (à partir de la description comportementale "algorithmes") commencent à apparaître sur le marché [BCU94] et existent depuis longtemps à l'état de prototypes universitaires [Gaj89] [GDW92] [JPO93].

Notons de plus que la simulation digitale concerne les représentations comportementales et structurelles des quatre niveaux, à l'exception de la représentation structurelle du niveau circuit en "transistors", qui est l'objet de la simulation électrique ou analogique.

3.1.2 Domaine analogique

En analogique, il est difficile d'établir une telle hiérarchie car le nombre de niveaux

dépend essentiellement de la complexité du système à réaliser. *Allen* [All86] propose une classification hiérarchique de la conception analogique en quatre niveaux: système, fonctionnel, circuit et composant (cf Tableau 2).

Tableau 2 Classification hiérarchique de la conception analogique de *Allen* [All86]

Niveau d'abstraction	Primitives de simulation	Unités de Conception
Système	Fonctions de transfert Domaine fréquentiel Domaine temporel	Architecture Topologie Connectivité
Fonctionnel	Équations algébriques linéaires et non-linéaires Courbes de transfert Tables	Sommateur, Intégrateur, Multiplieur, Bistable, Valeur absolue
Circuit	Macro-modèles: sources contrôlées éléments passifs dispositifs non-linéaires	Op Amps, Sources, Comparateurs
Composant	MOS, BJT, éléments passifs	Propriétés physiques ou géométriques, Tension, Courant

Il définit pour chaque niveau les "primitives de simulation" qui sont utilisées pour représenter le fonctionnement des "unités de conception" correspondantes. Ainsi, le plus bas niveau concerne les "composants" dont les primitives du simulateur (ou modèles) décrivent les propriétés physiques et géométriques. Un élément du niveau supérieur "circuit" est défini comme une interconnexion de composants et est représenté par un *macro-modèle*. Le niveau "fonctionnel" représente une combinaison de circuits afin de réaliser une fonction de traitement analogique, qui peut être décrite sous diverses formes: mathématiques, tables de valeurs, courbes de transfert. Enfin, le niveau "système" regroupe plusieurs "fonctions" interconnectées et est caractérisé par des fonctions de transfert (en s ou en z) étudiées dans les domaines temporels et fréquentiels.

Par rapport à la définition des niveaux d'abstraction du domaine digital du Tableau 1, cette classification ne met pas en valeur la dualité représentation comportementale et représentation structurelle de chaque niveau hiérarchique. Par exemple, les "circuits" tels que op-amps, comparateurs ou sources, utilisés au niveau "fonctionnel", sont modélisés par des macro-modèles qui constituent leurs "représentations comportementales" et sont

“structurellement” composés de transistors MOS ou BJT et d’éléments passifs. C’est cet aspect que tente de décrire le Tableau 3.

Tableau 3 Niveaux d’abstraction en analogique

Niveau d’abstraction	Représentation comportementale	Représentation structurelle
Système	Fonctions de transfert Schémas-blocs H(s), H(z) Domaine fréquentiel Domaine temporel Domaine Analogique/Digital	Convertisseurs A/D, D/A, PLL, Filtres, Sommateur, Intégrateur, Multiplieur, Bistable...
Fonctionnel	Équations algébriques linéaires et non-linéaires, Courbes de transfert, Tables	Op Amps, Sources, Comparateurs
Circuit	Macro-modèles	MOS, BJT, éléments passifs R,L,C
Composant	Modèles de composants	*Layout des composants*

On observe un décalage des “unités de conception” d’un niveau vers le haut. D’autre part, ce tableau permet de mieux appréhender les différents niveaux de la modélisation comportementale analogique et mixte. Des modèles de bas-niveau ou macro-modèles, basés sur les propriétés électriques, devront être écrits pour les circuits (op-amps, sources,...). Les descriptions pourront être plus abstraites aux niveaux supérieurs “fonctionnels” et surtout “système” où des systèmes mixtes doivent être considérés à l’heure actuelle.

3.2 Décomposition descendante

Pour illustrer le principe de cette méthodologie, nous prenons l’exemple de la conception d’une boucle à verrouillage de phase ou PLL selon le diagramme représenté en Figure 1 [MLN93]. Elle est constituée d’un détecteur de phase-fréquence (ou PFD) à pompe de charge, d’un filtre passe-bas, d’un oscillateur contrôlée par une tension (ou VCO) et enfin d’un compteur qui assure une division de la fréquence.

La conception débute par une description comportementale du système, sous forme mathématique ou algorithmique, qui sert de référence pour toute la conception. Dans le cas d’une PLL, ce modèle mathématique décrit la fonction de transfert de la boucle en utilisant la

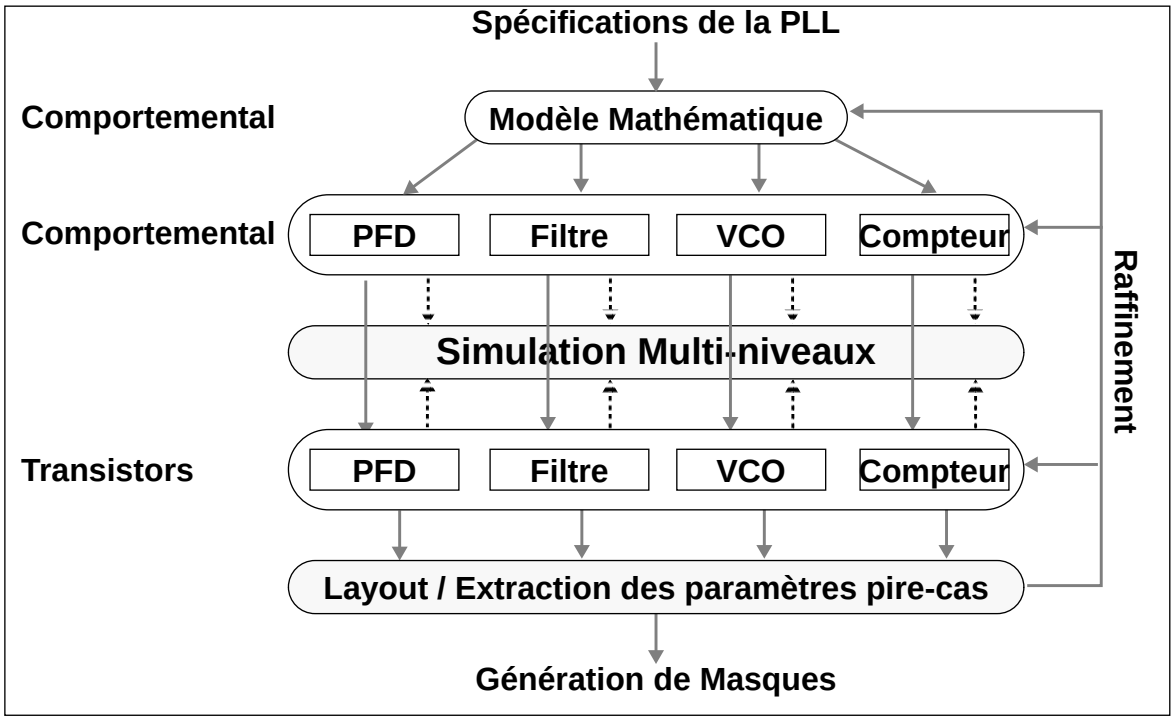


Figure 1 Flot de conception d'une PLL [MLN93]

variable de phase. Il est par exemple réalisé sous la forme d'un schéma-bloc représenté en Figure 2. Pour déterminer la fonction de transfert du PFD qui est analogique/digital, il faut considérer l'ensemble du détecteur et du filtre, ce qui conduit à la relation $\frac{U_0}{\Delta\Phi} = Kd \cdot Z(s)$, où *Z(s)* est l'impédance du filtre passe-bas [Gir91]. Quant au VCO, il est décrit par un simple intégrateur de gain *Ko* et le diviseur de fréquence est représenté par le bloc de gain *1/n*. Par une analyse fréquentielle, ce modèle permet de dimensionner les divers coefficients de la boucle afin de satisfaire les critères de stabilité.

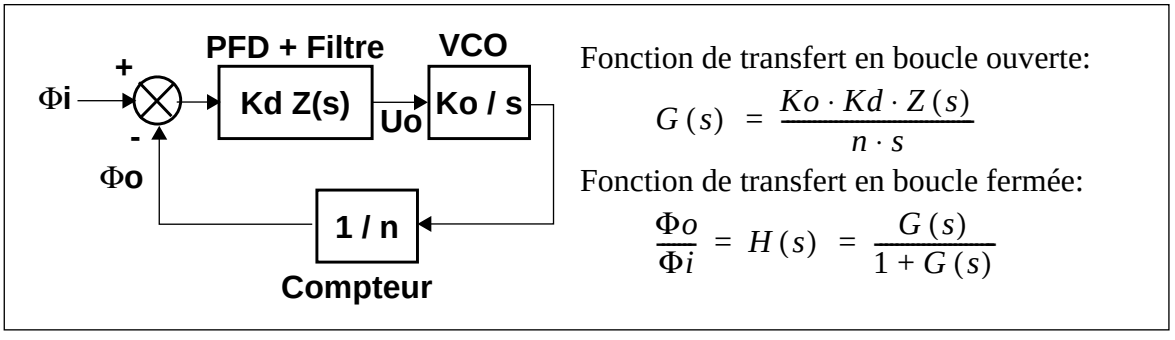


Figure 2 Modèle mathématique d'une PLL en variable de phase

Le système est ensuite décomposé en un certain nombre de blocs fonctionnels comportementaux et génériques. La simulation comportementale permet de définir les spécifications de chaque sous-bloc, afin que l'architecture choisie ait des performances

compatibles avec les spécifications du niveau supérieur. Si aucun jeu de paramètres adéquats ne peut être trouvé, l'architecture doit être modifiée. Des outils d'optimisation peuvent aussi être mis en oeuvre pour explorer de manière systématique l'espace de conception, en fonction de contraintes de performance mais aussi de coût. Dans le cas de la PLL, dont le schéma fonctionnel est décrit en Figure 3, on s'intéresse à ce niveau aux caractéristiques temporelles comme par exemple le temps de capture.

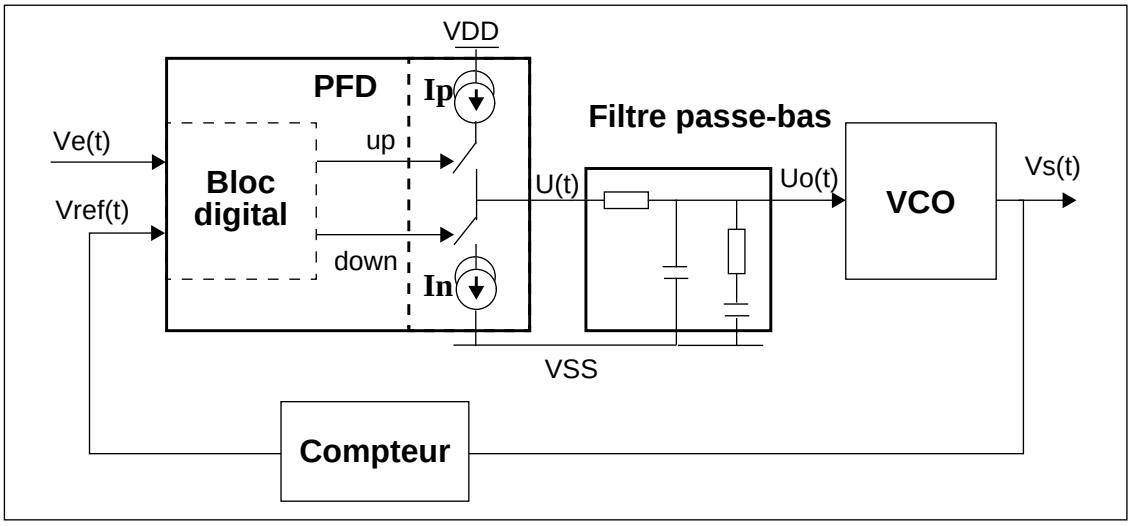


Figure 3 Schéma fonctionnel d'une PLL

La fonction du VCO est par exemple décrite par les équations suivantes:

- la phase est l'intégrale de la pulsation: $\varphi(t) = \int [\omega_0 + K_0 \cdot U_0(t)]\, dt$, où ω_0 représente la pulsation centrale du VCO, K_0 la sensibilité et $U_0(t)$ est la tension de contrôle,
- la tension de sortie est définie par $Vs(t) = a \cdot sin(\varphi(t))$. Elle est ensuite mise en forme de signal carré par une fonction de comparaison.

Le PFD est simplement décrit par une table d'états logiques, basée sur la détection des fronts montants des signaux d'entrée Ve(t) et de référence Vref(t) et générant deux signaux numériques (*up* et *down*) qui contrôlent deux interrupteurs analogiques constitutifs du bloc nommé *pompe de charge*. Trois états différents sont ainsi générés au noeud U(t) de la Figure 3: VDD, VSS et un état haute-impédance.

Le comportement temporel de la PLL, et en particulier le temps de capture, est étudié par une simulation comportementale très rapide: 25min de temps CPU sur une SPARCstation10, contre 3 jours si le circuit complet est au niveau transistor. Le signal Uo(t) est représenté en Figure 5: le temps de capture est de l'ordre de 0.7µs.

Après une telle étude fonctionnelle, la conception des schémas transistors de chaque bloc

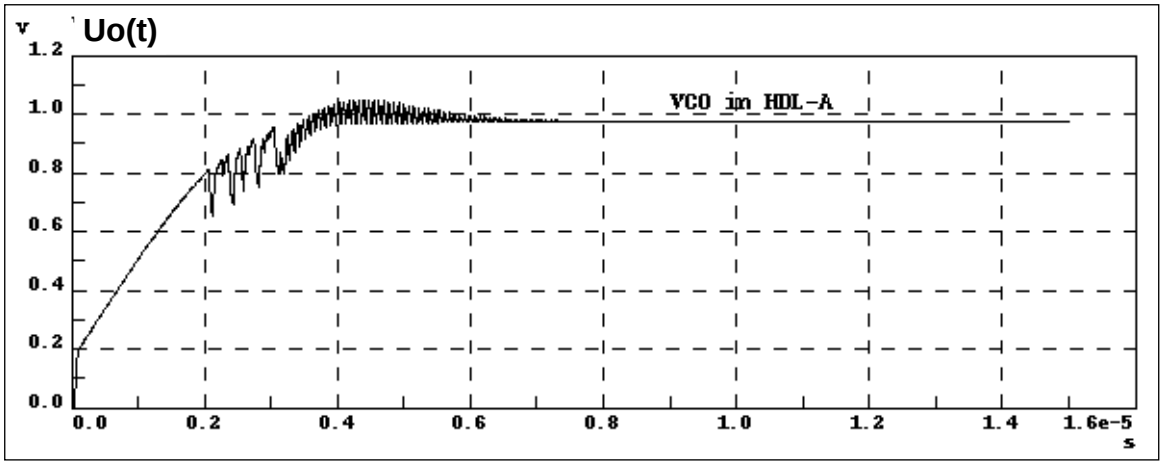


Figure 4 Sortie du filtre passe-bas de la PLL

peut alors débuter. Le nombre de niveaux hiérarchiques dépend bien sûr de la complexité du circuit. Pour la PLL, il est assez faible mais la Figure 5 présente un exemple beaucoup plus complexe décrit par *Liu et al.* [LSG91] et concernant la conception d'un circuit analogique/ digital codeur/décodeur, nommé CODEC. On remarque d'ailleurs qu'il comporte une PLL. Le terme de simulation comportementale désigne tout type de simulation mettant en jeu des modèles comportementaux. Ce peut être des études statistiques ou de bruit, effectuées par des simulateurs particuliers non nécessairement électriques.

Notons que pour un système mixte analogique-digital, le partitionnement des blocs fonctionnels entre les deux domaines doit intervenir le plus tôt possible afin d'utiliser les simulateurs digitaux, très efficaces, ainsi que les outils de synthèse logique. Cependant, le concepteur peut avoir à manipuler des blocs mixtes, à différents niveaux de description. Des langages de description mixte sont donc indispensables en particulier pour la modélisation de convertisseurs analogiques/digitaux [MaA90] [Rua91].

3.3 Validation ascendante

Dès qu'un bloc fonctionnel a été décrit de manière structurelle, une ***simulation multi-niveaux***, mettant en oeuvre non plus le modèle comportemental mais l'architecture de ce bloc, peut être effectuée. La Figure 1 du flot de conception d'une PLL met en évidence cette étape de simulation pour laquelle un seul bloc est décrit au niveau transistor alors que les autres blocs sont encore représentés de manière comportementale. Cela permet de vérifier très rapidement le fonctionnement du schéma dans son environnement réel pour, par exemple, résoudre les problèmes d'adaptation de charge. Ainsi, l'analyse temporelle du temps de capture de la PLL

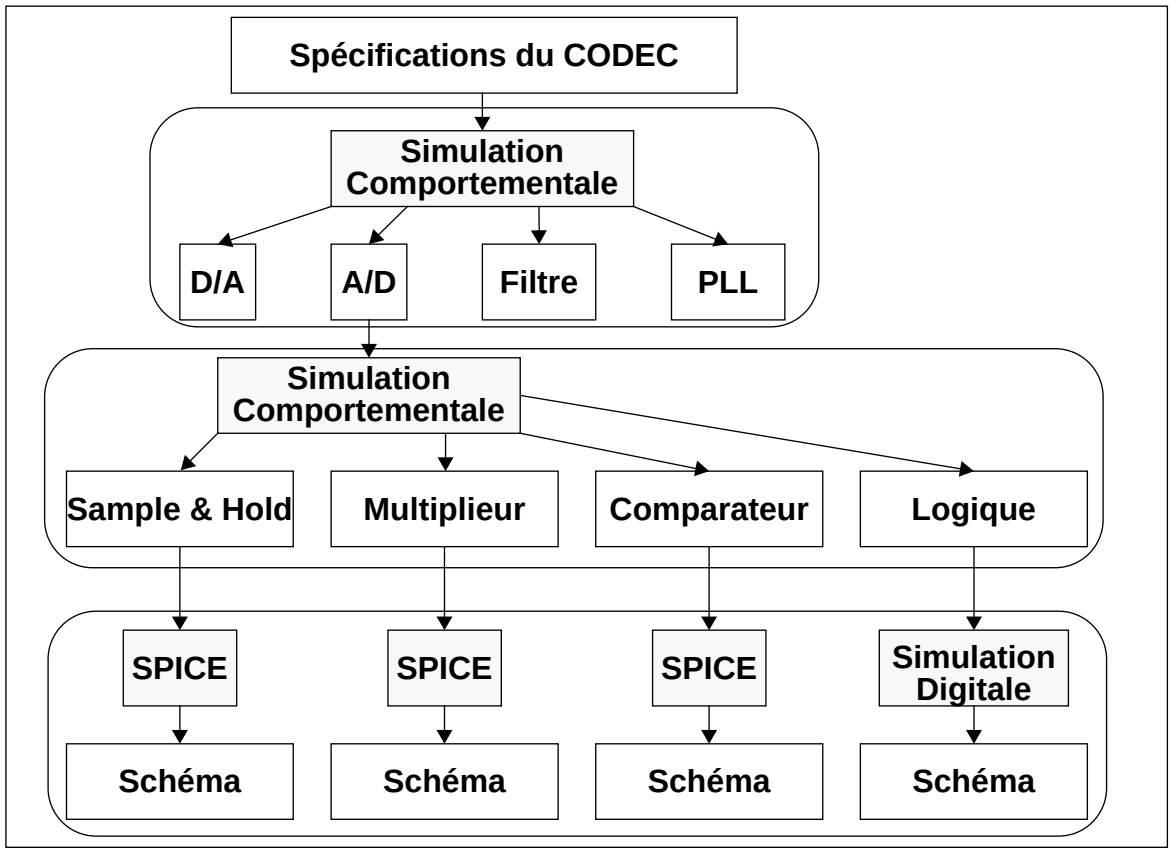


Figure 5 Hiérarchie dans un système analogique / digital [LSG91]

précédemment décrite, lorsque seul le VCO est au niveau schéma, ne coûte que 1h40min de temps CPU, ce qui reste acceptable.

D'autre part, les modèles comportementaux initialement conçus dans la phase de décomposition doivent être ***raffinés*** (cf Figure 1) en décrivant les spécificités de l'architecture réalisée et en particulier les phénomènes du second ordre. Concernant le VCO d'une PLL, la dépendance de la sensibilité vis à vis de la température est par exemple introduite [MLN93] ou encore la modélisation de non-linéarités qui sont sources d'harmoniques [LiS92]. De même, pour les convertisseurs analogiques/digitaux, il est essentiel d'introduire la description du bruit. Ceci est réalisé par *Liu et al.* [LCS93] à l'aide d'un formalisme statistique approprié. Une simulation comportementale à l'aide des nouveaux modèles plus précis doit alors être effectuée pour vérifier que les spécifications sont encore atteintes et pour évaluer dans quelles mesures les performances ont été modifiées. Ainsi, les problèmes de conception sont découverts au plus tôt

Ce raffinement des modèles peut aussi consister en la prise en compte de la *dispersion statistique* des paramètres des transistors, qui est à l'origine des phénomènes de différenciation

ou *mismatch* des paires de transistor des étages différentiels amplificateurs ainsi que des miroirs de courant [FNS94] [CLN94]. La modélisation de tels phénomènes permet de mieux apprécier certaines dégradations comme l'offset des amplificateurs et les erreurs de non-linéarité de certains convertisseurs analogiques/digitaux. De plus, afin de pouvoir effectuer des analyses statistiques de tolérance au niveau "système", *Koskinen at al.* [KoC92] élaborent des modèles comportementaux délivrant des informations statistiques: leurs paramètres (délais, pôle, slew-rate) sont en effet directement liés à des paramètres de transistors, par des fonctions polynômiales du premier ordre. La méthodologie *RSM* (*Response Surface Methodology*), basée sur une régression à partir d'un ensemble de résultats de simulations électriques, est ici utilisée.

Enfin, après réalisation du *layout*, une extraction des éléments parasites, capacitifs et résistifs, est effectuée. Là encore, les modèles peuvent être raffinés en incluant les phénomènes introduits de charge [LCS93]. Pour vérifier le respect des spécifications, la hiérarchie établie lors de la phase de décomposition descendante doit alors être parcourue de manière ascendante, en ajustant les paramètres des modèles comportementaux à chaque niveau, et en reportant les diverses variations jusque dans le modèle mathématique du plus haut niveau.

4 Simulation et diagnostic de fautes

L'utilisation des modèles comportementaux pour la testabilité et le diagnostic de fautes semble prometteuse [LKF94]. L'objectif est d'identifier la localisation des fautes et leur sévérité à partir des résultats de mesures. Les problèmes essentiels du test des circuits analogiques et mixtes reposent sur l'absence de consensus des modèles de fautes proposés et l'absence de simulateurs de fautes réellement efficaces. De ce fait, le test analogique, principalement basé sur la mesure de performances relatives à la fonctionnalité du circuit, est fortement redondant, malgré les travaux de *Milor et al.* [MiS90] pour établir un classement des tests à effectuer.

4.1 Modèles de fautes et simulation de fautes

Les modèles de fautes sont utilisés par la simulation de fautes pour calculer la couverture

de fautes d'une séquence de test, c'est à dire le pourcentage de fautes modélisées et détectées en appliquant tels stimuli. Cela permet donc de générer le test optimal qui couvre le maximum de fautes avec un nombre minimal de stimuli.

Les modèles de fautes doivent décrire les différents types de comportement erroné. Deux types de fautes sont généralement distingués dans le domaine analogique: les fautes catastrophiques (courts-circuits, circuits ouverts), qui provoquent généralement un dysfonctionnement complet, et les fautes paramétriques, dues à une dispersion des paramètres technologiques, et qui se révèlent par une variation des valeurs de certains composants (résistances, capacités) et la dégradation des performances par rapport à leur valeur nominale. *Marlett et al.* [MaA88] utilisent par exemple des résistances de shunt pour modéliser l'effet des court-circuits. Cependant de tels modèles de fautes ne permettent pas de traiter les circuits complexes. L'abstraction de la description des fautes est donc indispensable. *Meixner* [Mei93] a étudié l'élaboration de modèles de fautes pour des amplificateurs opérationnels: les résultats de simulation de différents types d'op-amp, dont le layout a été modifié pour modéliser des fautes, ont permis de déterminer un nombre réduit de classes de modèles. Pour chacune de ces classes, un macro-modèle assez simple a été décrit. Cela a permis de procéder à la simulation de fautes de circuits plus complexes (sample & hold, convertisseur).

Nagi et al. [NCA93] décrivent un simulateur de fautes basé sur la discrétisation des systèmes linéaires par transformation bilinéaire. Les principaux avantages sont la rapidité de simulation et la possibilité d'étudier les systèmes analogiques/digitaux. Les fautes modélisées se composent de courts-circuits, branches ouvertes, fautes paramétriques de composants passifs et des modèles comportementaux de fautes d'amplificateurs. Ces derniers modélisent les changements de gain, de tension d'offset, de slew-rate et le décalage des pôles.

4.2 Diagnostic de fautes

Le but du diagnostic de fautes est de déterminer la localisation précise et la sévérité des fautes détectées par les mesures.

Une approche basée sur la théorie des circuits et la résolution de systèmes d'équations décrivant le réseau, a été proposée par *Liu* [Liu91]. Les variations des paramètres du circuit y sont définies à partir de la mesure des perturbations des signaux accessibles et la localisation des erreurs est déterminée par la détermination du rang d'une matrice qui dérive de la matrice

d'admittance du système. Cette méthode est exploitée par *Walker et al.* [WAL92] qui utilisent la modulation de certains éléments du réseau.

Une autre approche consiste en l'analyse des sensibilités de la fonction de transfert et des performances par rapport aux différents paramètres du circuit [SlK92]. La mesure des variations relatives des performances permet d'évaluer les variations relatives des composants. Là encore, la complexité du circuit impose l'utilisation de descriptions abstraites. Dans [LKF94], la matrice des sensibilités est calculée à l'aide de simulations basées sur un modèle comportemental. Un nouvel algorithme permet de résoudre les problèmes dûs à la corrélation des vecteurs de sensibilité, pour finalement obtenir une estimation aux moindres carrés des erreurs des composants.

5 Conclusion

Ce chapitre a permis de mettre en valeur les buts de la modélisation comportementale: réduire les temps de simulation et permettre l'étude de systèmes complexes analogiques/ digitaux, améliorer la qualité de la conception par application de la méthodologie de conception hiérarchique descendante (top-down) associée à une validation ascendante (bottom-up), et même faciliter la simulation et le diagnostic de fautes.

On peut d'autre part distinguer deux classes de modèles fonctionnels selon les phases de conception:

- les modèles ***génériques*** qui sont utilisés lors des étapes de décomposition hiérarchique et décrivent la fonction nominale idéale et certaines non-idéalités qui sont critiques pour le choix de l'architecture du système, telles que les temps de propagation, la distorsion, le bruit, etc... Ils sont de plus indépendants de toute architecture et technologie.
- les modèles ***de validation*** qui représentent un circuit (ou bloc) particulier et sont construits après réalisation du layout. Leurs paramètres sont obtenus à partir de la caractérisation du circuit réel. Ils peuvent tenir compte des fluctuations statistiques du procédé et sont, dans ce cas, bien adaptés à l'optimisation du rendement, ainsi qu'au diagnostic de fautes. Un compromis entre précision et vitesse de simulation doit cependant être effectué.

Chapitre 3: Méthodologies de modélisation

1 Introduction

Alors que le chapitre précédent présentait les applications des modèles comportementaux pour résoudre les problèmes de simulation et de conception des circuits analogiques/digitaux actuels, celui-ci vise à recenser les principales méthodologies d'élaboration. Les propriétés des divers langages disponibles, structurels ou comportementaux, sont tout d'abord décrits. Sont ensuite présentées les méthodes couramment utilisées pour la définition des équations et des paramètres des modèles comportementaux analogiques.

2 Les différents types de langages

Deux familles de langages peuvent être distinguées: les langages de bas-niveau, choisis pour des raisons d'efficacité numérique dans le codage des primitives des simulateurs, et les langages dédiés à la description de matériel ou HDL (*Hardware Description Language*), tels que VHDL dans le domaine digital.

2.1 Les langages de bas-niveau

Ces langages (FORTRAN, C) sont utilisés pour coder les primitives des simulateurs électriques: ce sont des langages efficaces pour les calculs numériques, mais difficilement utilisables par un concepteur de circuits.

Cependant, pour faciliter l'écriture du code C de nouveaux modèles, des outils ont été développés (par exemple pour les simulateurs ADVICE [VCC88], ADAMS [AnE92], [SiS91]). Ils permettent en particulier de générer un système d'équations d'états à partir de la simple spécification d'une fonction de transfert de Laplace. Dans [VCC88], un outil de

vérification “ACME” est présenté. Son but est d’étudier le modèle qui a été écrit dans le langage C. “ACME” évalue le modèle indépendamment du simulateur, afin d’analyser sa précision et la continuité des équations. Cette méthode permet de garantir la fonctionnalité du modèle avant toute simulation.

Dans certains cas, le modèle décrit grâce à un langage de haut niveau de type VHDL, est directement traduit en C avant sa compilation. C’est le cas du langage analogique décrit dans [MSS92] et associé au simulateur *iMACSIM*. En plus de la vérification syntaxique, la continuité des fonctions et celle des dérivées est analysée pour les besoins du simulateur. Un lissage des discontinuités peut même être effectué et les équations susceptibles de provoquer des erreurs numériques sont détectées.

Enfin, certains simulateurs offrent des bibliothèques de fonctions permettant ainsi à l’utilisateur d’écrire ses propres modèles C. C’est le cas du simulateur Eldo qui propose un ensemble de fonctions C dédiées à la modélisation analogique et appelé CFAS (*C Functional Analog Simulation*) [Cfa93]. Le code compilé des nouveaux modèles doit être archivé dans une bibliothèque qui sera liée au simulateur à l’exécution.

2.2 Les langages de description de matériel (HDL)

Ce sont des langages spécialement conçus pour la description de systèmes, électroniques ou autres. Ils constituent l’interface d’entrée des simulateurs, des outils de preuve formelle et de synthèse, et peuvent contribuer à faciliter la spécification et la documentation de circuits. Trois types de description doivent être envisagés [ABO90]:

- la ***description structurelle***, qui donne des informations sur la structure des blocs et composants utilisés. Le langage d’entrée du simulateur SPICE est de ce type. Dans une hiérarchie de description, les éléments décrits de cette façon ne peuvent constituer des éléments terminaux. Cependant, nous verrons dans le paragraphe suivant que la description structurelle est utilisée pour un certain type de description comportementale, la macro-modélisation.
- la ***description comportementale***, qui exprime le fonctionnement du bloc, c’est à dire ses équations, sans se soucier de sa structure interne. Un langage de description purement comportementale, comme le langage de modélisation analogique FAS [Fas92] associé au simulateur Eldo, est distinct d’un langage de programmation classique dans la mesure où il

manipule de nouveaux types de données selon des lois adaptées à la description physique des composants. En électronique analogique, le langage doit permettre l'application des lois de Kirchhoff, qui correspondent, pour d'autres domaines physiques, à d'autres lois de conservation de l'énergie. De même dans le domaine digital, l'affectation des signaux de VHDL vérifie les principes de causalité grâce à la notion de delta-délai et propose différentes règles de propagation [ABO90].

- la ***description flot de données***, qui est définie par VHDL pour exprimer rapidement les flots de données sortant du modèle en fonction des flots entrants, sans référence à la structure interne.

Le Tableau 4 tente d'établir un classement des langages cités dans cette thèse, par type de représentation et type de circuit: MAST [Mas93] associé au simulateur Saber [Sab87], FAS [Fas92], CFAS [Cfa93] et HDL-A [Hdl94] associés à Eldo [Eld94], les standards SPICE, VHDL, et enfin le standard en cours de définition VHDL-A [DOD94] [SRC94].

Tableau 4 Classement des langages abordés dans cette thèse

Domaine	Représentation structurelle	Représentation comportementale
Digital	VHDL	VHDL
Analogique/Digital	MAST HDL-A (future version) VHDL-A	MAST HDL-A (version étudiée) VHDL-A
Analogique	SPICE	FAS, C-FAS

Le standard ***VHDL-A*** ou ***IEEE 1076.1*** définit les extensions analogiques du standard digital VHDL 1076 et doit permettre la description de systèmes mixtes analogiques-digitaux pouvant appartenir à différents domaines physiques: systèmes électriques, mécaniques, thermiques... Notons que le langage HDL-A a été conçu en suivant les principaux objectifs de VHDL-A et il sera considéré, dans le cadre de ce rapport, comme une version intermédiaire ou encore un prototype du futur standard.

Citons aussi le futur standard ***MHDL*** (*MIMIC HDL*) dédié à la description et simulation de circuits micro-ondes analogiques et mixtes [SRC94]. C'est un nouveau langage, indépendant de VHDL. L'interface de ce langage avec un ensemble d'outils CAO concernant entre autre la synthèse analogique et l'automatisation des tests, est défini dans le même temps.

Signalons que MHDL n'a pas été étudié au cours de cette thèse.

Ces travaux de standardisation sont très importants pour l'échange de modèles entre les compagnies industrielles, la construction de bibliothèques de modèles et le développement de nouveaux outils CAO tels que les outils de synthèse.

3 Utilisation d'un langage structurel (Macro-Modélisation)

Ce paragraphe montre comment un langage de description structurelle, tel que le langage d'entrée du simulateur SPICE, nommé lui-même SPICE par simplification, peut être utilisé pour la modélisation comportementale de fonctions analogiques à la place d'un langage comportemental. Cette méthode de modélisation est couramment appelée ***macro-modélisation***.

Comme nous l'avons déjà vu au chapitre précédent, le réseau électrique, décrit en langage SPICE, est analysé par le simulateur afin de construire un système d'équations, basé sur les équations de Kirchhoff et les équations des composants. La macro-modélisation consiste soit en la ***construction d'un schéma*** qui conduira aux relations souhaitées entre des variables représentées par les tensions de noeud et les courants de branche, soit en la ***simplification d'un schéma*** afin de réduire le nombre de noeuds du circuit initial.

Nous présentons ci-après les quelques composants qui constituent les briques de base des macro-modèles: les sources contrôlées, les éléments réactifs et les diodes.

3.1 Les sources contrôlées

Ce sont des éléments idéaux qui permettent d'exprimer facilement des relations mathématiques entre tensions et courants. Quatre sources contrôlées polynômiales sont par exemple disponibles dans les version actuelles de SPICE:

- source de tension contrôlée par des tensions (VCVS), de la forme $v = e(v_1, \ldots, v_n)$,
- source de tension contrôlée par des courants (CCVS), de la forme $v = h(i_1, \ldots, i_n)$,
- source de courant contrôlée par des tensions (VCCS), de la forme $i = g(v_1, \ldots, v_n)$,
- source de courant contrôlée par des courants (CCCS), de la forme $i = f(i_1, \ldots, i_n)$.

L'ordre des polynômes est paramétrable ainsi que les coefficients. Outre les fonctions algébriques d'addition, de soustraction et de multiplication, ces polynômes peuvent aussi décrire une série de Taylor [CoC92] et approximer ainsi de nombreuses autres fonctions mathématiques (sinusoïdales, exponentielles ou logarithmiques).

Cependant, les coefficients des polynômes doivent être des constantes numériques. Pour décrire des fonctions paramétrables, dans le but d'effectuer par exemple une famille de simulations, les paramètres variables doivent être calculés électriquement et introduits sous la forme de nouvelles tensions de contrôle [Her93]. Mais cela peut augmenter considérablement le nombre de noeuds du réseau, ce qui doit être évité.

Les simulateurs plus récents, tels que Eldo, possèdent en fait des sources plus évoluées dont la valeur peut être exprimée directement soit par une table de valeurs de type $(x_i, y_i)_{i=1\ldots n}$, soit par une expression algébrique faisant appel à une bibliothèque étendue de fonctions mathématiques. Des possibilités de calcul des divers paramètres sont aussi offertes, ce qui facilite l'écriture de fonctions complexes. Cependant, chaque simulateur possède sa propre syntaxe, ce qui freine l'utilisation de ces fonctions si le but est l'échange des macro-modèles.

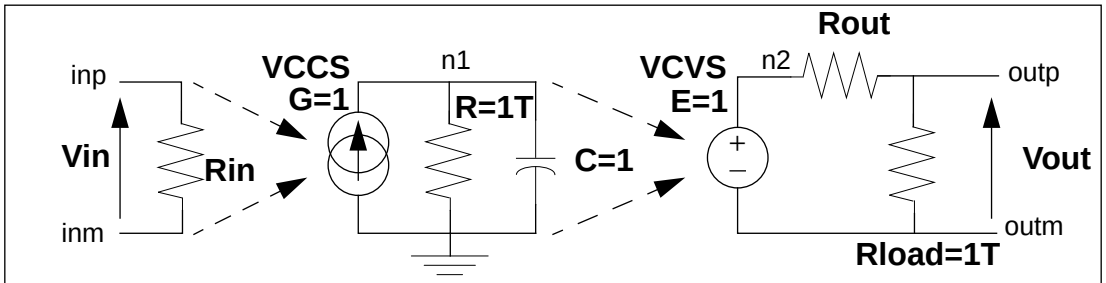
3.2 Les éléments réactifs

Capacités et inductances sont utilisées pour réaliser des opérateurs de dérivation et d'intégration qui sont ensuite utilisés pour décrire des fonctions de transfert en s.

La Figure 6 présente l'exemple d'un module ***intégrateur*** constitué d'un étage d'entrée d'impédance Rin, d'un étage intermédiaire représentant un filtre passe-bas et enfin d'un étage de sortie d'impédance Rout. La fonction de transfert du filtre $H(s) = \frac{V_{int}}{V_{in}} = \frac{G \cdot R}{1 + R \cdot C \cdot s}$ se réduit à $H(s) = \frac{1}{s}$ en choisissant une résistance R assez grande (1 TΩ par exemple), une transconductance G de 1, et une capacité C de 1 F. Les étages d'entrée et de sortie permettent en fait d'isoler l'étage principal des éléments extérieurs. La résistance de charge de 1TΩ de la sortie autorise l'utilisation du macro-modèle sans résistance de charge extérieure.

Ce module intégrateur, associé à des blocs multiplieurs et des sommateurs, sert de brique de base pour la construction de fonctions de transfert de Laplace plus complexes [Her93]. Remarquons enfin l'utilisation de la commande `.SUBCKT`, commune à tous les simulateurs de type SPICE et qui permet de définir ainsi un bloc ou *sous-circuit*. Ce modèle est de plus

paramétré par les résistances d'entrée et de sortie, `rin` et `rout`.




```
.SUBCKT integ inp inm outp outm PARAM: rin=1K rout=1
  Rin inp inm rin
  Gint 0 n1 1
  Rint n1 0 1T
  Cint n1 0 1
  Eout n2 outm n1 0 1
  Rout n2 outp rout
  Rload outp outm 1T
.ENDS integ
```

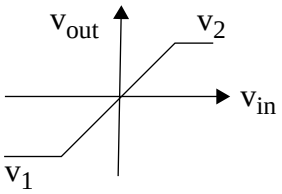
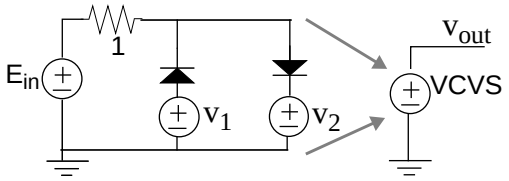
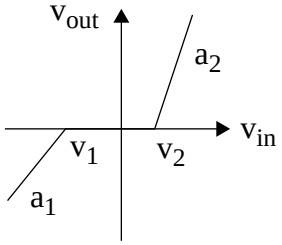
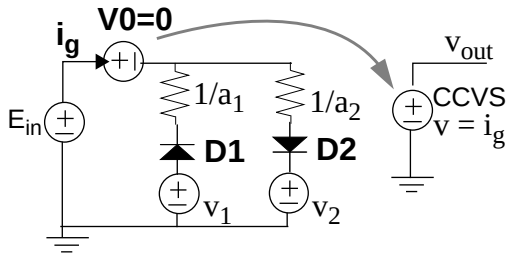
Figure 6 Macro-modèle d'un intégrateur pour Eldo

3.3 Les diodes

La caractéristique non-linéaire des ***diodes*** est utilisée en tant qu'opérateur conditionnel dans des comparateurs [Cal91], des limiteurs de tension, des modèles d'amplificateurs à "zone morte" (dead zone) et plus généralement pour la génération de fonctions linéaires par morceaux.

Le Tableau 5 présente des exemples simples du limiteur de tension et de l'amplificateur "zone morte", qui ont une structure similaire, et où la source VCVS E_{in} est contrôlée par la tension d'entrée v_{in}. Alors que pour le limiteur de tension, la grandeur transmise en sortie est la tension aux bornes des diodes, dans le cas de l'amplificateur *zone-morte*, il s'agit du courant i_g qui est mesuré à travers la source de tension nulle `V0`. Ce courant varie selon la conduction de D1 et D2 et les résistances 1/a1 et 1/a2 permettent de définir les gains a1 et a2: lorsque Vin est inférieur à V1, seule D1 conduit et le courant est égal à $i_g = a_1 \cdot (v_{in} - v_1)$; pour Vin compris entre V1 et V2, D1 et D2 sont bloquées; enfin, lorsque Vin est supérieur à V2, D2 seule conduit et le courant est égal à $i_g = a_2 \cdot (v_{in} - v_2)$.

Tableau 5 Limiteur de tension et amplificateur "zone-morte"

Caractéristique	Schéma
Limiteur de tension: 	
Amplificateur zone morte (dead-band): 	

Cependant, les diodes introduisent un décalage de tension qui est exprimé par la relation: $V = N \cdot \frac{k \cdot T}{q} \cdot ln\left(\frac{I}{I_S} - 1\right)$, où *N* est le coefficient d'émission, *Is* le courant de saturation et *I* le courant traversant la diode. Trois méthodes sont généralement employées pour réduire ce phénomène parasite:

- le décalage de tension est compensé par une source de tension fixe en série avec la diode, en faisant l'hypothèse d'un courant maximal [BCP74],
- les variables sont systématiquement multipliées par un facteur d'échelle (de l'ordre de 1000) de telle façon que la tension de seuil des diodes devienne négligeable,
- le coefficient d'émission est diminué, de 1 à 0.001 par exemple, selon les facilités de convergence du simulateur [CoC92].

3.4 Avantages et inconvénients

Cette technique de macro-modélisation ne nécessite donc pas l'apprentissage d'un langage de programmation mais requiert simplement une bonne connaissance d'un simulateur analogique et des primitives disponibles, ainsi que des techniques classiques de l'électronique (réalisation de filtres, boucles de rétro-action, schémas de diodes,...). Elle permet ainsi de réaliser un grand nombre de fonctions (fonctions mathématiques, filtres, détecteurs de pics, opérations sur les fréquences, détecteurs de phase, modulateurs d'amplitude et de largeur

d'impulsions, oscillateur, convertisseur fréquence-tension, comparateurs, amplificateurs et boucle à verrouillage de phase, etc...), décrites par exemple dans [CoC92]. C'est une méthode populaire qui a donné lieu à un grand nombre d'études.

Cependant, malgré sa simplicité apparente, la macro-modélisation pose un certain nombre de problèmes:

- des *parasites* sont introduits par certains composants tels que les diodes, qui possèdent une tension de seuil non négligeable,
- des problèmes de *convergence* sont parfois à résoudre lors de l'utilisation de boucles de rétro-action [Cal91],
- la *paramétrisation* est difficile, du moins pour les premiers simulateurs SPICE. Par exemple, les coefficients des sources contrôlées polynômiales sont des constantes. Une solution consiste alors à introduire les paramètres comme des tensions de contrôle. Une approche plus conviviale est de générer automatiquement la description structurelle des modèles par un programme de pré-traitement qui calcule les valeurs des composants en fonction des paramètres définis par l'utilisateur [CoC92]. Enfin, les nouveaux simulateurs offrent de plus grandes facilités de paramétrisation et de calcul d'expressions mathématiques. Ainsi, pour Eldo [Eld94], la valeur des paramètres d'un grand nombre de composants peut être définie par une relation entre accolades {} en fonction des paramètres du modèle.
- la *plage de fonctionnement* est généralement assez *limitée* du fait de la difficulté de paramétrisation,
- la macro-modélisation est limitée aux *circuits analogiques*.

4 Utilisation d'un langage comportemental

Étudions maintenant les fonctions indispensables à la modélisation comportementale de systèmes analogiques-digitaux. Cette réflexion s'inscrit dans le cadre du suivi de la standardisation en cours de VHDL-A (IEEE 1076.1), langage qui étend VHDL à l'analogique [DOD93] [Vac93] [SRC93]. Aussi, le langage HDL-A est ici pris comme référence car il est très proche du futur standard. Notons de plus que les principaux problèmes de modélisation

rencontrés au cours de ces travaux sont ici exposés.

4.1 Systèmes multi-disciplinaires

Un langage de description comportementale analogique doit supporter la description de systèmes ***multi-disciplinaires*** et en particulier des ***microsystèmes*** [RLR95] [DCB95], tels que des micro-capteurs, micro-pompes, micro-valves ou micro-moteurs, qui comportent des fonctions électriques, thermiques, mécaniques, optiques ou magnétiques. En effet, pour simuler de tels systèmes, des modèles comportementaux doivent être élaborés pour décrire les composants "physiques" qui interagissent fortement avec les parties électriques. Par exemple, l'étude de la régulation ou asservissement électronique d'un moteur doit absolument tenir compte de l'influence de la charge mécanique sur ses propriétés électriques. De même, les phénomènes thermiques, pouvant influer fortement sur le comportement des différents composants, sont à prendre en compte dans la modélisation d'un microsystème.

Les applications d'un langage comportemental analogique sont en fait liées aux caractéristiques du simulateur utilisé et plus particulièrement à la formulation des équations du réseau. Ainsi, les simulateurs électriques de type SPICE ne peuvent traiter que des systèmes d'équations différentielles ordinaires en fonction du temps du fait de l'approximation des régimes quasi-permanents qui est effectuée. En particulier, les algorithmes de ces simulateurs ne sont pas adaptés aux circuits micro-ondes et plus généralement aux systèmes distribués, nommés aussi à constante répartie, qui devront être "discrétisés" en première approximation [ARN70].

D'autre part, les *lois de Kirchhoff* des tensions et des courants sur lesquelles est basé tout simulateur électrique, sont issues des lois de conservation de l'énergie et peuvent donc être généralisées à d'autres domaines ***mécaniques***, ***thermiques***, ***hydrauliques***, ***pneumatiques*** obéissant aussi à des principes de conservation. On les nommera dans la suite ***lois de Kirchhoff généralisées*** [SRC94]. De même, tensions et courants ont leurs équivalents dans les autres domaines. On définit de manière générale deux types de variables pour chaque domaine physique:

- le type ***potentiel***: la valeur des variables de ce type est propre à chaque noeud du réseau et est définie par rapport à un noeud de référence (un pour chaque domaine). Le terme anglais est ***across***.

- le type ***flux***, nommé ***through*** en anglais: il désigne les grandeurs relatives à une branche comprise entre deux noeuds. Un flux est d'autre part une grandeur *absolue* et *unidirectionnelle*.

Le Tableau 6 présente les couples de grandeurs qui sont généralement choisis pour décrire les systèmes électriques, mécaniques et thermiques, ainsi que les lois associées de connexion des éléments du système. On trouvera en Annexe 2 des exemples de systèmes mécaniques, thermiques et hydrauliques.

Tableau 6 Quelques domaines physiques

Domaine	"Potentiel"	"Flux"	Loi de connexion aux noeuds du réseau
Électrique	Tension V (volt)	Courant I (A)	Loi de Kirchhoff $\sum_{noeud} i = 0$
Mécanique Translation	Déplacement X (m) ou Vitesse V (m/s)	Force F (N)	Loi de la dynamique $\sum_{solide} F = M \cdot \frac{d^2x}{dt^2} = M \cdot \frac{dV}{dt}$
Mécanique Rotation	Déplacement θ (rad) ou Vitesse Ω (rad/s)	Couple Γ (Nm)	Loi de la dynamique $\sum_{solide} \Gamma = J \cdot \frac{d^2\theta}{dt^2} = J \cdot \frac{d\Omega}{dt}$
Thermique	Température (K)	Flux de chaleur (J/s ou Watt)	Conservation du flux de chaleur
Hydraulique	Pression (Pa) Hauteur Liquide (m)	Débit Vol. (m^3/s)	Conservation du débit
Pneumatique	Pression (Pa)	Débit massique (kg/s)	Conservation du débit

D'autre part, pour la description de systèmes multi-disciplinaires, le langage de modélisation doit assurer la cohérence des grandeurs et en particulier leur point de référence et leurs unités. Ainsi, VHDL-A propose de créer un nouvel objet, nommé ***nature***, qui permettra à l'utilisateur de définir divers domaines physiques, caractérisés par les noms des variables de type potentiel et flux et par le nom du noeud de référence [SRC93].

Par exemple, le domaine *electrical* du langage HDL-A est défini par:

```
TYPE electrical IS NATURE
     ACROSS v: ANALOG;  -- potentiel
     THROUGH i: ANALOG; -- flux
     REFERENCE gnd;     -- référence
```

```
END NATURE electrical;
```

Précisons que ces facilités de modélisation multi-disciplinaire sont essentielles pour l'utilisation d'un langage de modélisation comportementale dans le cadre du développement d'environnements de CAO de microsystèmes.

4.2 Les connexions analogiques

Deux types de modèles comportementaux sont envisageables et sont associés, chacuns, à un type particulier de connexion analogique:

- les *modèles de bas niveau* pour lesquels les "impédances" d'entrée/sortie doivent être prises en compte, sont caractérisés par des interconnexions "physiques" qui obéissent aux lois de Kirchhoff généralisées,
- les *modèles de haut-niveau*, utilisés dans des *schémas-blocs*, sont liés entre eux par des grandeurs mathématiques qui n'appartiennent pas nécessairement à un domaine physique particulier.

4.2.1 Connexions "physiques"

Les modèles de bas-niveau expriment des ***relations de branche*** entre les diverses bornes de connexion. Or, chaque borne analogique est caractérisée par deux grandeurs:

- le ***potentiel*** du noeud auquel elle est connectée,
- le ***flux*** entrant par ce point dans la branche correspondante du modèle.

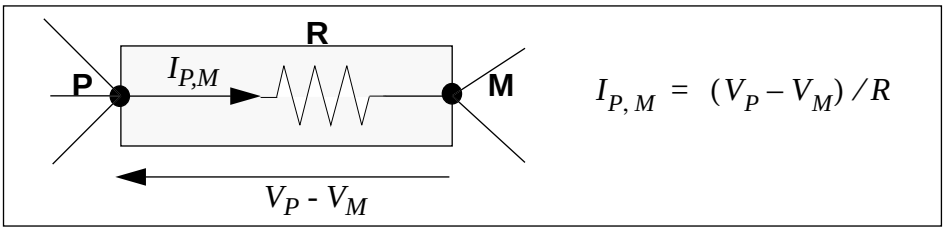
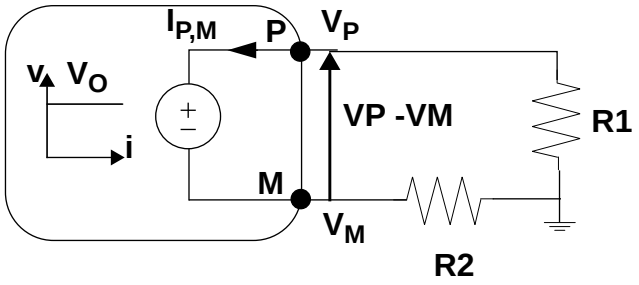


Figure 7 Modèle d'une résistance linéaire R

Prenons tout d'abord l'exemple d'une ***résistance*** R de bornes *P* et *M* (cf Figure 7), appartenant au domaine électrique défini au paragraphe 4.1. La relation de branche correspond à la loi d'Ohm $I_{P,M} = (V_P - V_M)/R$, où $I_{P,M}$ est le courant de branche et V_P et V_M les potentiels des deux points de connexion. La formulation HDL-A de ce modèle est de la forme: `[P,M].i %= [P,M].v/R;` où:

- `[P,M].i` désigne le courant de la branche P,M dans le sens P vers M,
- `[P,M].v` correspond à la différence de potentiel entre P et M,
- `%=` est un opérateur d'affectation signifiant l'application de la loi de Kirchhoff des courants (ici en P et M) et permettant de distinguer les ***relations de connexion*** du modèle.

Dans le cas d'une ***source idéale de tension***, le courant traversant la source est déterminé par le simulateur en fonction de la topologie du circuit. C'est une inconnue du simulateur et du modèle lui-même. La Figure 8 montre comment ce courant est calculé sur un exemple précis. Pour une source de bornes P et M et de valeur V0, la formulation proposée par HDL-A est simplement: `[P,M].v %= V0;`



Système à 3 inconnues (selon M.N.A.) $I_{P,M}$, V_P et V_M et 3 équations:

(1) KCL en P: $I_{P,M} + \frac{V_P}{R_1} = 0$

(2) KCL en M: $I_{P,M} - \frac{V_M}{R_2} = 0$

(3) Source de tension: $V_P - V_M = V_o$ ou relation de contrainte.

Figure 8 Mise en équation d'un réseau comportant une source idéale de tension

Cependant, l'accès au courant circulant dans une source de tension peut être nécessaire. C'est par exemple le cas d'une source avec fusible interne, dont l'état dépend du niveau de courant: s'il est supérieur à un seuil donné (**Imax**), le courant s'annule et la tension aux bornes du composant devient une inconnue qui dépend de la topologie du circuit. D'une source de tension d'inconnue le courant, on passe donc à une source de courant d'inconnue la tension. La Figure 9 correspond au diagramme fonctionnel de ce modèle où la variable `Iout` correspond au courant de la branche **[P,M]**. En HDL-A, cette propriété est en fait exprimée par la relation de connexion définissant une source de courant: `[P,M].i %= Iout;`

Lorsque `Iout<Imax`, le modèle correspond à une source de tension qui doit être définie

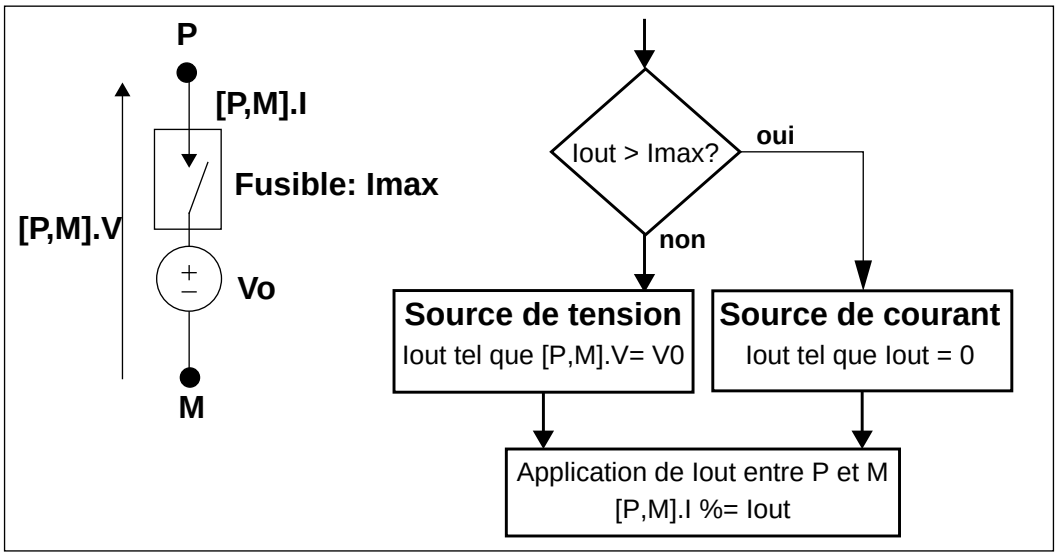


Figure 9 Source idéale de tension avec fusible de seuil Imax

par une relation de contrainte. Pour accéder à la valeur de `Iout`, HDL-A requiert l'utilisation d'une ***équation implicite*** qui est résolue par le simulateur de la même façon que les autres équations du système. Formellement, elle s'écrit:

```
résoudre Iout tel que [P,M].V == V0;
```

La syntaxe HDL-A est précisée au Chapitre 5.

Signalons une autre approche proposée par MAST [Mas93] qui évite l'utilisation d'une équation implicite en déclarant au préalable pour chaque branche un couple de variables *différence de potentiel* et *courant* (cf Annexe 9). Le simulateur déduit de lui-même des équations quelle variable de branche est l'inconnue.

4.2.2 Connexions "mathématiques"

Les modèles de haut-niveau sont définis comme des blocs fonctionnels utilisés par exemple dans des schémas-blocs pour étudier la stabilité de systèmes de régulation électrique ou mécanique ou autre. Les grandeurs analogiques manipulées sont donc plutôt considérées comme des variables mathématiques et il n'y a pas lieu de soumettre les connexions entre blocs aux lois de Kirchhoff. Signalons que de telles connexions sont aussi établies par les quatre sources contrôlées idéales du simulateur SPICE qui permettent de lire soit des tensions en des noeuds de contrôle, soit des courants circulant à travers des sources de tension ou des résistances, sans introduire de perturbations.

Il est donc nécessaire de disposer d'un autre type de connexion, semblable au ***couplage***

(***coupling***) de flux entre deux inductances mutuelles (cf Figure 10): le flux magnétique de chaque inductance couplée dépend à la fois du courant qui circule dans ses spires et du courant circulant dans l'autre inductance. Le modèle d'une inductance couplée doit donc avoir trois points de connexion: deux bornes électriques et un couplage.

La connexion de couplage possède d'autre part une *direction*: la variable transmise par ce biais est exportée d'un modèle qui en définit la valeur et importée dans un modèle qui ne fait que lire cette valeur.

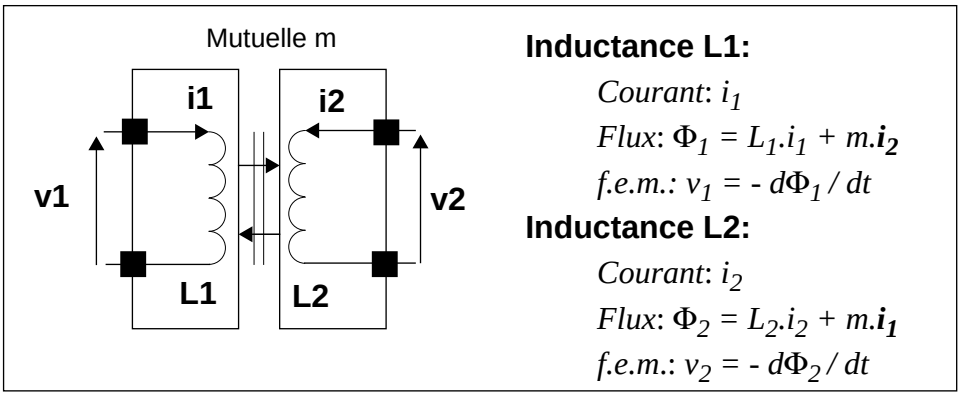


Figure 10 Équations d'inductances couplées $m = \sqrt{L_1 \cdot L_2}$

4.3 Domaines d'interprétation

Un langage comportemental doit d'autre part supporter les principales analyses qui sont utilisées pour l'étude des circuits analogiques, à savoir le calcul du point de fonctionnement en régime permanent ou analyse *DC*, l'analyse temporelle ou *transitoire* et enfin l'analyse fréquentielle en petits-signaux ou *AC* (cf Chapitre 2). Les simulations de bruit et statistiques seraient aussi aussi à prendre en compte.

À chaque forme d'analyse correspond généralement un ensemble spécifique d'équations. Un modèle doit donc être sub-divisible en plusieurs blocs appelés ***domaines d'interprétation*** dans les documents de standardisation de VHDL-A [Vac93]. Par exemple, les équations DC peuvent servir à l'initialisation de certaines variables et de certains noeuds du circuit. Alors que la description des éléments réactifs y est inutile, la représentation temporelle doit par contre en tenir compte. Lorsque celle-ci est basée sur des équations différentielles, le simulateur doit être capable d'en déduire automatiquement la description fréquentielle (après détermination du point de fonctionnement autour duquel le système d'équations est linéarisé). Par contre, dans le cas de la modélisation de circuits échantillonnés, comme la PLL décrite au

Chapitre 2, les représentations fréquentielles et temporelles sont totalement différentes: la description mathématique fréquentielle met en effet en jeu la variable de phase au lieu des tensions/courants qui sont étudiés lors des simulations temporelles.

De même, les variables analogiques possèdent des représentations différentes selon le mode d'analyse, temporel ou fréquentiel. En effet, lors d'une *analyse temporelle*, les variables analogiques sont des grandeurs réelles, discrétisées par le simulateur et qui sont représentées par une suite de points $(t_i, x_i)_{i = 1 \ldots n}$ en un certain nombre de pas de temps. Les valeurs entre deux pas de temps peuvent être déterminées par interpolation linéaire. Notons que la modélisation de délais requiert la sauvegarde des valeurs précédemment calculées pendant un intervalle de temps donné. Lors d'une *simulation fréquentielle*, ces variables appartiennent par contre au domaine complexe et sont déterminées à chaque fréquence de l'analyse.

Pour prendre en compte cette notion de polymorphisme, un type spécifique doit être défini, nommé par exemple ***analog*** en HDL-A.

4.4 Opérateurs analogiques requis

4.4.1 Dérivation

L'opérateur de dérivation en fonction du temps est indispensable pour décrire le comportement des composants réactifs, inductances et capacités ainsi que des fonctions de transfert de Laplace du type:

$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{a_0 + a_1 \cdot s + \ldots + a_n \cdot s^n}{b_0 + b_1 \cdot s + \ldots + b_m \cdot s^m}$, qui peut se mettre sous la forme:

$$b_0 \cdot V_{out} + b_1 \cdot s \cdot V_{out} + \ldots + b_m \cdot s^m \cdot V_{out} = a_0 \cdot v_{in} + a_1 \cdot s \cdot V_{in} + \ldots + a_n \cdot s^n \cdot V_{in}$$

Dans le domaine *temporel*, cette fonction est décrite par l'équation différentielle suivante:

$$b_0 \cdot v_{out} + b_1 \cdot \frac{dv_{out}}{dt} + \ldots + b_m \cdot \frac{d^m v_{out}}{dt^m} = a_0 \cdot v_{in} + a_1 \cdot \frac{dv_{in}}{dt} + \ldots + a_n \cdot \frac{d^n v_{in}}{dt^n}$$

Rappelons que les grandeurs v_{in} et v_{out} sont complexes lors de l'analyse AC et réelles lors d'une étude temporelle. De plus, la représentation *fréquentielle* peut être obtenue automatiquement par le simulateur à partir de la description temporelle, en remplaçant les dérivées n-ièmes de V par $(j \cdot \omega)^n \cdot V$.

De façon générale, une équation différentielle est modélisée par une *équation implicite* d'inconnue la tension de sortie v_{out}. Cependant, lorsque le coefficient b_0 n'est pas nul, une autre approche peut être envisagée en calculant de manière *explicite* une grandeur analogique *vs* qui est appliquée sur la borne de sortie:

$$v_s = \left(a_0 \cdot v_{in} + a_1 \cdot \frac{dv_{in}}{dt} + \ldots + a_n \cdot \frac{d^n v_{in}}{dt^n} - \left(b_1 \cdot \frac{dv_{out}}{dt} + \ldots + b_m \cdot \frac{d^m v_{out}}{dt^m} \right) \right) / b_0$$

Dans cette démarche, la tension v_{out} est au préalable lue sur cette même borne de sortie. Cela revient en fait à définir une équation de contrainte $v_{out} = v_s$ dans le système d'équations du simulateur.

4.4.2 Intégration

Pour la modélisation d'un intégrateur pur $H(s) = \frac{1}{s}$, la disponibilité d'un opérateur d'intégration permet d'exprimer directement la tension de sortie v_{out} par la relation intégrale $v_{out} = v_0 + \int v_{in} d\tau$, où v_0 désigne la constante d'intégration. On peut cependant se passer d'un tel opérateur en écrivant l'équation différentielle associée $\frac{dv_{out}}{dt} = v_{in}$ à l'aide d'une équation implicite d'inconnue v_{out} dans le domaine temporel et d'une initialisation $v_{out} = v_0$ en DC.

Ces deux types de formulation sont comparés dans le Tableau 7 concernant la modélisation d'une capacité et d'une inductance. On remarquera que les descriptions DC et temporelles de la formulation différentielle sont les mêmes car en régime permanent les dérivées s'annullent. Ce n'est pas la cas de la formulation intégrale pour laquelle une équation implicite est nécessaire en DC pour calculer les constantes d'intégration.

Tableau 7 Modélisation d'une capacité et d'une inductance

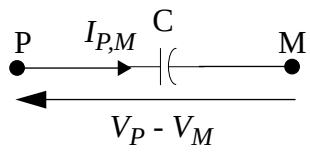
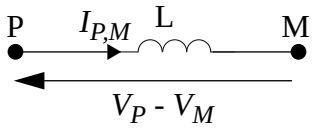
Composant	Formulation Différentielle	Formulation Intégrale
Capacité  	***Équation DC et temporelle*** $I_{P,M} = C \times \frac{d}{dt}(V_P - V_M)$	***Équation DC:*** définition de V_0 $V_0 = V_P - V_M$ tel que $I_{P,M} = 0$ ***Équation temporelle:*** $I_{P,M}$ est tel que $V_M - V_P = \int_0^t (I_{P,M}/C)\, dt + V_0$

Tableau 7 Modélisation d'une capacité et d'une inductance

Composant	Formulation Différentielle	Formulation Intégrale
Inductance  	***Équation DC et temporelle*** $I_{P,M}$ est tel que $V_P - V_M = L \times \frac{dI_{P,M}}{dt}$	***Équation DC:*** définition de I_0 I_0=$I_{P,M}$ tel que V_P - V_M = 0 ***Équation temporelle:*** $I_{P,M} = \int_0^t (V_{P,M}/L)\,dt + I_0$

4.4.3 Délai

Cet opérateur est indispensable à la modélisation des fonctions de transfert en z des filtres digitaux [TiS91] du type:

$$H(z) = \frac{V_{out}(z)}{V_{in}(z)} = \frac{a_0 + a_1 \cdot z^{-1} + \ldots + a_n \cdot z^{-n}}{1 + b_1 \cdot z^{-1} + \ldots + b_m \cdot z^{-m}}$$

Sa définition est exposée à l'aide de la Figure 11 dans les domaines temporels et fréquentiels.

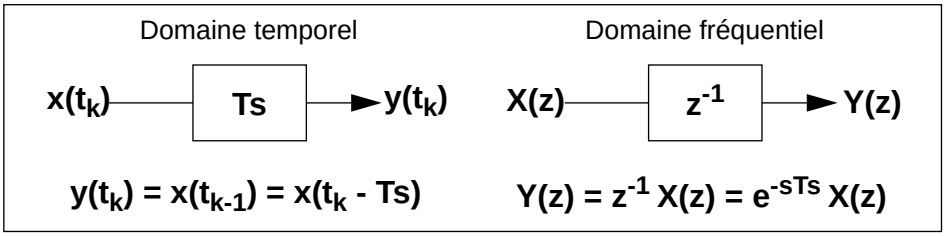


Figure 11 Représentation d'un élément délai de retard Ts [TiS91]

Dans le domaine fréquentiel, cet opérateur est donc complexe. Dans le domaine temporel, il retourne une valeur déterminée à un certain instant du passé. Signalons que la fonction de transfert *H(z)* précédente peut être modélisée dans le *domaine temporel* en utilisant la formule récurrente suivante:

$$v_{out}^{k} = a_0 \cdot v_{in}^{k} + a_1 \cdot v_{in}^{k-1} + \ldots + a_n \cdot v^{k-n} - \left(b_1 \cdot v_{out}^{k-1} + \ldots + b_m \cdot v_{out}^{k-m} \right)$$

où k est l'indice d'échantillonnage et $V^k = V(t_k)$ est un échantillon calculé à chaque multiple de la période *Ts* en fonction des précédents: $V^{k-m} = V(t_{k-m}) = V(t_k - m \cdot T_s)$.

4.4.4 Détection d'évènements analogiques

Pour décrire un comparateur ou un convertisseur A/D, il est nécessaire de détecter avec

précision le franchissement des seuils de quantification par le signal analogique d'entrée. La Figure 12 présente le comportement d'un comparateur à hystérésis de seuils V_1 et V_2 et de niveaux hauts et bas: *Vdd* et *Vss*.

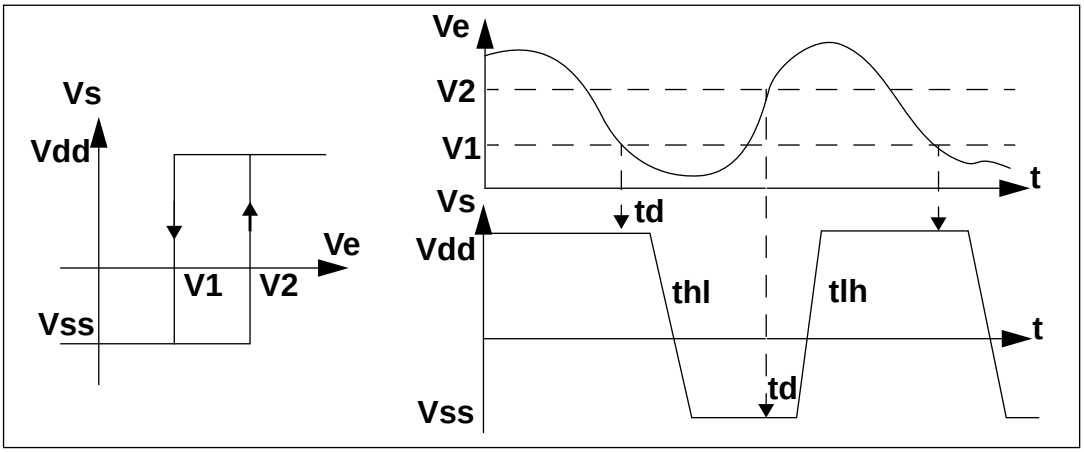


Figure 12 Modèle idéal du comparateur à hystérésis

L'état initial du comparateur est déterminé par la description DC en fonction du niveau de la tension d'entrée par une simple instruction conditionnelle (formelle):

```
IF  Ve < V2  THEN  Vs = Vss  ELSE  Vs = Vdd
END IF
```

La description temporelle tient compte du phénomène d'hystérésis et utilise l'opérateur délai pour connaître l'état précédent. La description formelle est la suivante:

```
IF  Vs(t − Δt) = Vdd  AND ( Ve(t − Δt) > V1 AND  Ve(t) < V1 ) THEN
        Vs(t) = Vss
ELSE
       IF  Vs(t − Δt) = Vss  AND ( Ve(t − Δt) < V2  AND Ve(t) > V2 ) THEN
             Vs(t) = Vdd
       END IF
END IF
```

Δt désigne l'intervalle de temps entre l'instant présent et le point précédent et $V(t-\Delta t)$ la valeur rendue par l'opérateur délai de paramètre Δt appliqué à la variable analogique *V*. L'expression booléenne ($Ve(t-\Delta t) > V1$ AND $Ve(t) < V1$) permet ainsi de détecter le franchissement descendant par *Ve* du seuil V_1. Des fonctions spécialisées sont proposées pour une description plus concise, telles que RISING() et FALLING() en HDL-A.

Notons que la précision de la détection dépend fortement du pas de temps Δt du simulateur au moment du passage de seuil. L'instant exact peut être approché par interpolation mais il apparaît important de pouvoir *limiter* ce pas de temps selon le degré de précision

souhaité.

4.4.5 Génération d'ondes analogiques

Pour la modélisation de sources impulsionnelles, triangulaires, exponentielles ou sinusoidales, ainsi que pour la génération de transitions caractérisées par un délai et un temps de commutation non nul, l'accès au temps interne du simulateur et le contrôle de son pas de temps sont des fonctionnalités indispensables. Précisons que la programmation d'évènements peut être facilement réalisée de manière digitale, dans le cadre d'un langage de modélisation mixte.

5 Définition des équations

Nous avons déjà cité quelques techniques permettant de générer des équations: il s'agit des méthodes d'approximation *AWE* et d'analyse symbolique qui sont intéressantes pour déterminer des fonctions de transfert de circuits linéarisés (cf Chapitre 2).

Deux autres approches peuvent être envisagées. La première correspond à la construction ***modulaire*** de modèles génériques et paramétrables qui sont décomposés en un ensemble de blocs fonctionnels élémentaires interconnectés. La seconde consiste en une ***simplification*** du schéma "transistors" du circuit étudié. Le modèle obtenu est dans ce cas lié à une structure particulière de circuit mais reste néanmoins paramétrable.

5.1 Construction modulaire

Une telle méthodologie est décrite par *Connelly et al.* [CoC92] pour la macro-modélisation à l'aide du langage SPICE, ainsi que dans de nombreuses autres références concernant la macro-modélisation d'op-amps [ChL75], [AlB90] ou [KeM94]. Elle est en fait générale et s'applique également à la modélisation par langage comportemental. Un diagramme de blocs fonctionnels est ainsi construit pour décrire le comportement du modèle. Chaque bloc est, si possible, dédié à la description d'une caractéristique particulière.

Les sections de base d'un bloc fonctionnel sont représentées par le diagramme de la Figure 13 qui comporte deux étages d'E/S de part et d'autre d'un étage central décrivant la fonction de transfert principale.

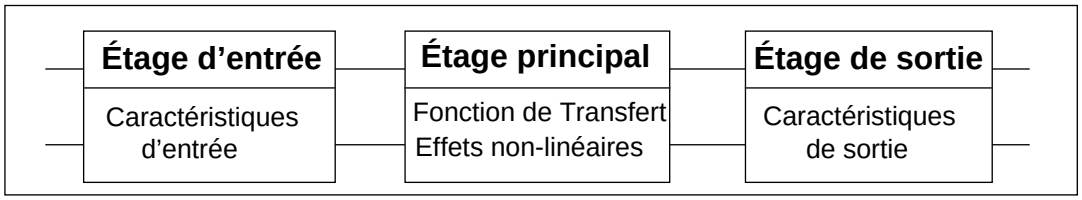


Figure 13 Sections de base d'un bloc fonctionnel [CoC92]

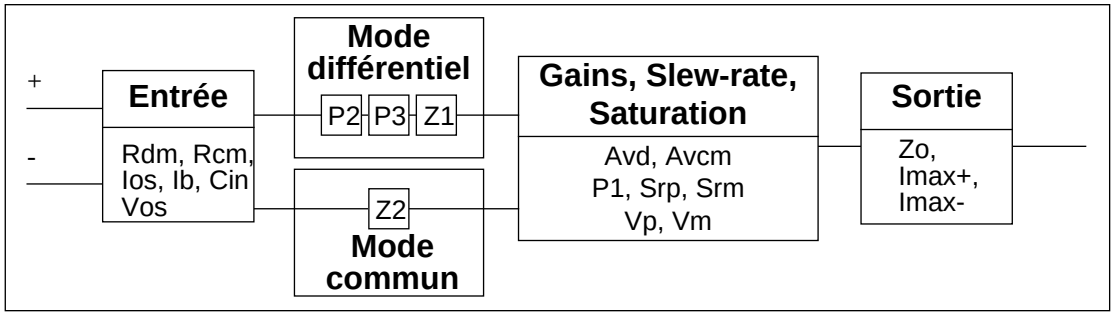


Figure 14 Schéma fonctionnel d'un macro-modèle d'Op-Amp d'après [CoC92]

La Figure 14 donne l'exemple du schéma-bloc d'un macro-modèle d'amplificateur, plus complexe, composé des étages suivants:

- un étage différentiel d'entrée modélisant les résistances différentielles (Rdm) et de mode commun (Rcm), la capacité d'entrée (cin), les courants de polarisation et de décalage (Ib et Ios) et enfin la tension de décalage (Vos),
- un ensemble d'étages décrivant les pôles et zéros secondaires de la fonction de transfert différentielle (P2, P3, Z1), ainsi qu'un zéro dans la fonction de transfert de mode commun (Z2),
- un étage modélisant le gain différentiel (Avd), le gain de mode commun (Avcm) et le pôle dominant (P1), les limitations des tensions positives et négatives (Vp et Vm) et enfin les *slew-rates* positifs et négatifs (Srp et Srm),
- un étage de sortie modélisant l'impédance de sortie (Zo) et la limitation du courant qui peut être délivrée lorsque l'amplificateur est saturé à Vp (resp. Vm): Imax+, (resp. Imax-).

On distingue d'autre part deux types de connexion entre blocs fonctionnels:

- la plus courante consiste à considèrer les blocs comme des *modules indépendants* les uns des autres. Dans le cadre de la macro-modélisation, ils sont connectés entre eux par l'intermédiaire de sources contrôlées pour éviter tout effet de charge, comme pour le macro-modèle d'intégrateur présenté dans le paragraphe 3.2 du Chapitre 3.

- certaines fonctionnalités doivent cependant être regroupées en un seul bloc: c'est le cas de la modélisation du gain, du pôle dominant, de la limitation du slew-rate et de tension pour le macro-modèle de *Connelly et al.*, comme nous le verrons au Chapitre 5. Disons simplement ici que le slew-rate y est décrit par limitation du courant de charge de la capacité associée au pôle dominant de l'op-amp. De même, la limitation de tension est associée au gain pour éviter la génération de tensions physiquement aberrantes conduisant à des résultats erronnés dans certaines conditions de fonctionnement.

5.2 Simplification de circuits

Les équations d'un modèle peuvent d'autre part être déterminées par analyse du réseau électrique selon une approche de simplification, c'est à dire de réduction du nombre de noeuds et de remplacement de la plupart des composants non-linéaires par des éléments idéaux (sources contrôlées, résistances, capacités). Selon cette méthode, la structure des étages qui sont essentiels pour le comportement du circuit est conservée et le modèle obtenu est donc relativement précis.

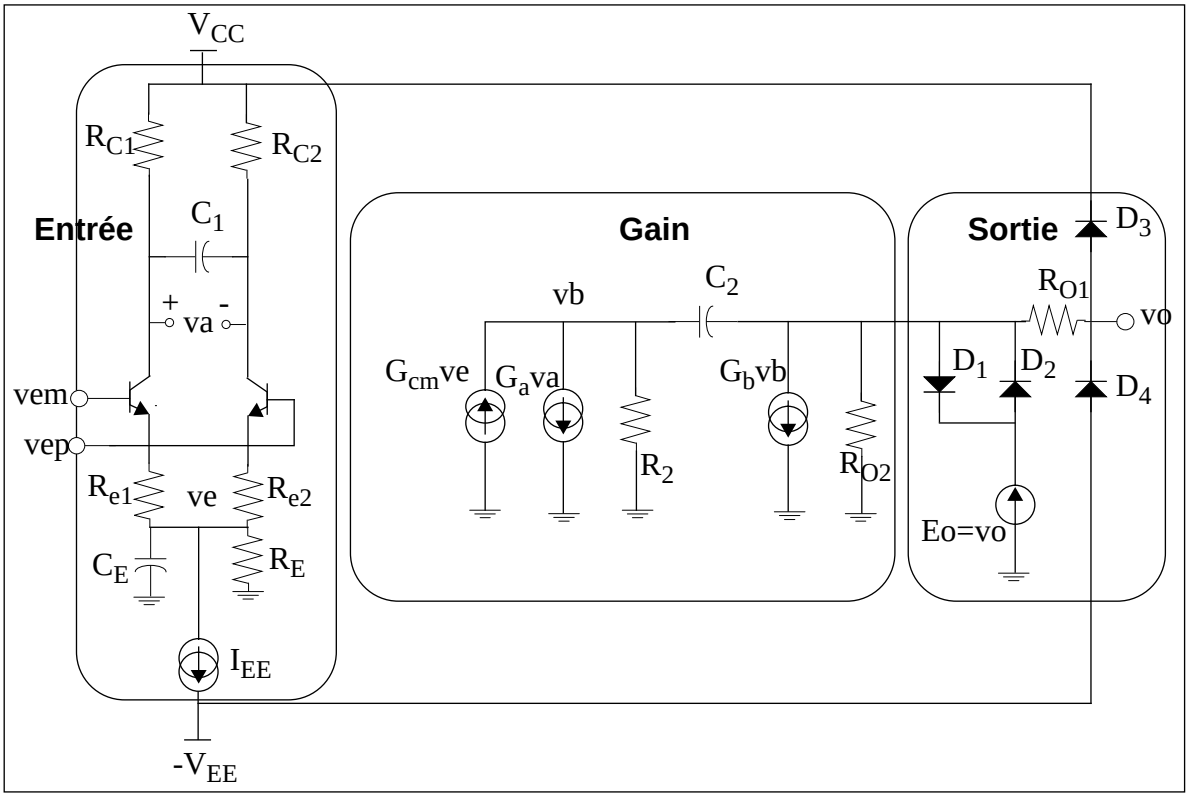


Figure 15 Macromodèle d'OP-AMP [BCP74] constitué de trois étages

Le macro-modèle d'amplificateur opérationnel décrit par *Boyle et al.* [BCP74] est en partie conçu selon cette méthodologie. Il est représenté en Figure 15 et comporte des étages d'entrée, de gain et de sortie. La technique de simplification a été utilisée pour les deux premiers étages. Ainsi, l'étage différentiel d'entrée est basé sur une paire de transistors idéaux dont les paramètres, totalement indépendants de la technologie, sont ajustés de manière à modéliser la tension de décalage et les courants de polarisation. On retrouve d'autre part la capacité de compensation de l'amplificateur (C_2), qui est à l'origine du pôle dominant. Concernant l'étage de sortie, des blocs fonctionnels de limitation de tension et de courant ont été simplement employés car la méthode de simplification conduit dans ce cas à un nombre de noeuds supérieur.

Une automatisation de cette approche de simplification est proposée par *Mantooth et al.* [MaA93] pour la macro-modélisation de comparateurs. Ils décrivent un algorithme permettant de classer les noeuds d'un circuit selon leur degré de contribution à ses performances statiques et dynamiques. Ne sont physiquement représentés que les noeuds parcourus par le signal traité et connectés à des composants responsables des principaux pôles ou zéros. Les noeuds influant sur les caractéristiques grands-signaux, comme par exemple le noeud commun à une paire différentielle de transistors, sont aussi conservés. Des sources contrôlées de courant sont aussi utilisées pour modéliser les relations associées à des noeuds de moindre importance.

Le modèle obtenu est très précis et prend en compte de nombreux phénomènes du second ordre, telles que la dépendance des racines de la fonction de transfert vis à vis de la polarisation.

6 Génération des paramètres

Dans le cadre de la phase de conception de validation ascendante, les macro-modèles ou modèles comportementaux doivent posséder des performances les plus proches possibles de celles des circuits qu'ils représentent. Leurs paramètres sont déterminés par *caractérisation* des propriétés du circuit, à l'aide de simulations et de l'analyse des courbes obtenues. Les *data-sheets* ainsi générées peuvent être ensuite utilisées pour calculer analytiquement des paramètres de macro-modèles. D'autre part, pour pouvoir effectuer des *analyses statistiques*, des méthodes existent qui relient les paramètres comportementaux à certains paramètres

transistors. Enfin, des algorithmes d'*optimisation* sont requis pour minimiser les différences de performances entre circuit et modèle.

6.1 Outils de caractérisation

La caractérisation d'un circuit peut avoir lieu à tous les niveaux d'une hiérarchie de conception après réalisation d'une architecture et surtout après définition du *layout* et extraction des éléments parasites. La caractérisation des circuits analogiques est plus complexe que pour les portes logiques car de nombreuses performances doivent généralement être analysées. Cela nécessite en pratique plusieurs types de simulations (DC, AC, transitoires) généralement effectuées sur différents schémas de mesure (*test-benches*).

Un certain nombre d'outils ont été développés pour programmer les analyses et traiter ensuite les courbes obtenues pour le calcul de performances. Le Tableau 8 cite les principaux et la Figure 16 présente le principe d'une caractérisation à l'aide de *Simboy.*

Tableau 8 Outils de caractérisation

Produit	Applications	Langage de commande
AnaCar [HGT91]	Développement de bibliothèques ASIC analogiques/digitales	C-Lang: proche du C
Simboy [NHM94] [HHN94]	Plans de caractérisation et de synthèse	Tableur avancé, fonctions C
Pluto [Plu92]	Optimisation du rendement, Caractérisation d'op-amp [ACT92]	Pluto: proche du C-Shell

Les caractéristiques communes de ces divers outils sont les suivantes:

- chaque type de mesure est considéré comme une *procédure paramétrée* par les valeurs des stimuli, de certains composants, par la température, les paramètres de commande du simulateur... Ces paramètres peuvent être des résultats d'autres procédures de mesure comme dans le cas de l'analyse AC d'un op-amp en boucle ouverte: le schéma de caractérisation nécessite de placer, en entrée de l'op-amp, une source de tension qui compense sa tension de décalage. Celle-ci est obtenue par une analyse DC.
- la description ou *netlist* du circuit étudié peut être elle aussi paramétrée.
- une bibliothèque de fonctions de traitement des courbes est proposée pour évaluer des propriétés complexes: gain, pôles, zéros, marge de phase, distorsion...

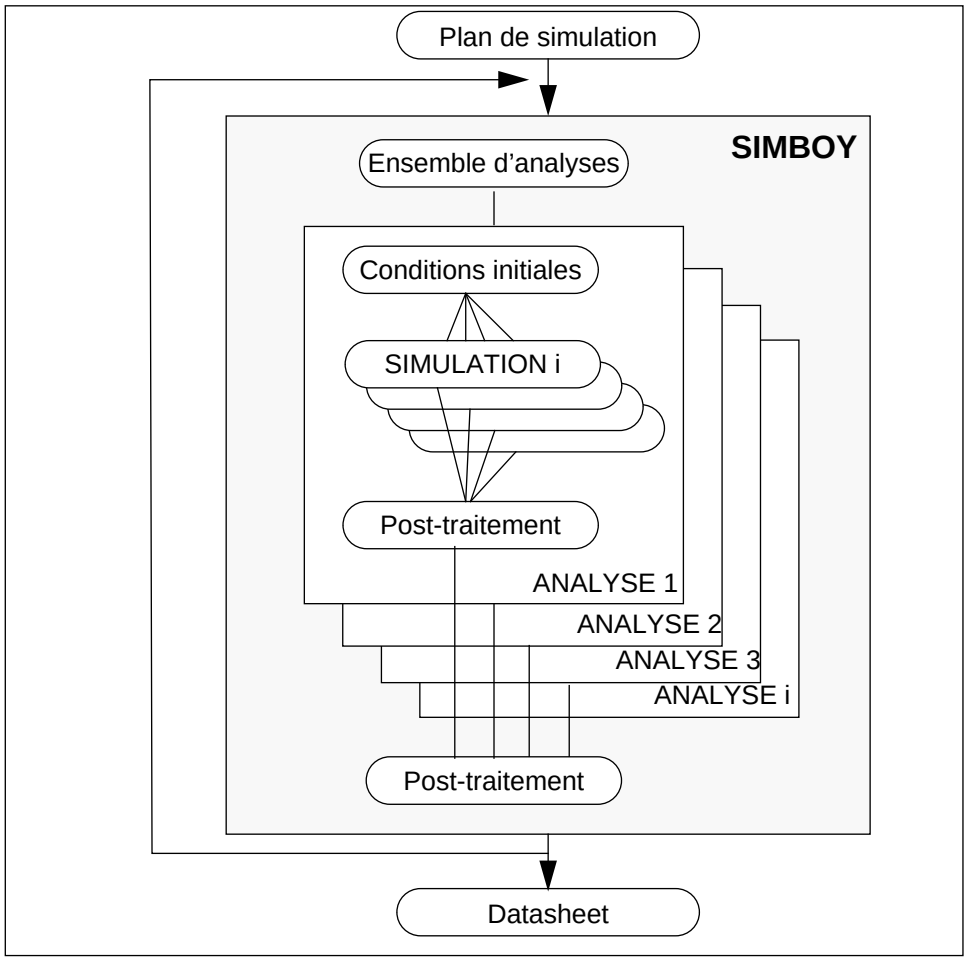


Figure 16 Caractérisation utilisant des plans de simulation [NHM94]

- des fonctions de contrôle de la simulation et de la caractérisation sont définies. Ainsi, des itérations peuvent être indispensables pour rechercher la plage d'analyse permettant de mesurer correctement les performances souhaitées [ACT92]: c'est par exemple le cas de l'extraction du slew-rate (cf Chapitre 6).
- des fonctions de manipulation de fichiers (lecture, écriture) sont aussi proposées.

6.2 Caractérisation statistique

L'élaboration de modèles comportementaux permettant d'analyser les tolérances des performances au niveau système et de calculer le rendement du circuit, connaissant les lois statistiques de dispersions des composants, revêt une importance primordiale [GSB93]. La méthodologie *RSM* (*Response Surface Methdology*) est mise en oeuvre dans ce but par *Koskinen et al.* [KoC92] ainsi que par l'outil *Pluto* précédemment cité [Plu92]. Chaque performance considérée (offset, gain, bande-passante, slew-rate, délai) peut être ainsi

approchée par régression, sur un domaine limité, par un polynôme du premier ou du second degré. Ce dernier est fonction de paramètres transistors caractérisés par une distribution statistique et éventuellement corrélés. Notons que l'utilisation par *Pluto* de plans d'expérience permet de définir les jeux de valeurs des paramètres du circuit conduisant à une régression optimale avec un nombre réduit de simulations. Ces polynômes, appelés par ailleurs macro-modèle, sont évalués très rapidement en remplacement des simulations lors des études statistiques de Monte-Carlo, qui requièrent toujours un très grand nombre de tirages aléatoires.

6.3 Optimisation

L'optimisation est indispensable pour minimiser les différences entre les performances ou les courbes de sortie du circuit et celles du modèle associé. Par exemple, *Casinovi et al.* [CaS91] et *Ju et al.* [JRS91] s'intéressent à l'optimisation des performances dynamiques d'un macro-modèle d'op-amp et en particulier à la modélisation du slew-rate et du pôle dominant. Dans ces deux références, le macro-modèle étudié est basé sur celui de *Chua et al.* [ChL75], dont l'étage de gain est représenté en Figure 17: les degrés de liberté consistent en la transconductance non-linéaire *gm*, caractérisée par des pentes éventuellement différentes et par le courant maximal Im, la résistance *R* (éventuellement non-linéaire) et la capacité *C*. Précisons que les fonctions linéaires sont décrites par des tables de valeurs (*look-up tables*).

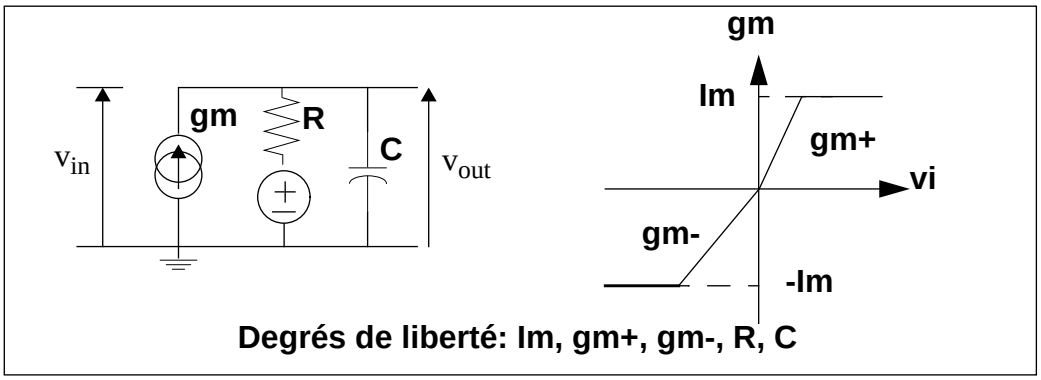


Figure 17 Macro-modèle d'op-amp étudié pour l'optimisation [CaS91] [JRS91]

Pour ce problème d'optimisation, deux approches différentes sont proposées: le calcul des sensibilités et les simulations de Monte-Carlo.

La méthode décrite par *Casinovi et al.* est ainsi basée sur le calcul des sensibilités dans le domaine temporel de la fonction de coût par rapport aux divers paramètres du modèle et des stimuli. Cette étude se résume en fait au problème *min-max* suivant: le meilleur macro-modèle

est celui qui minimise la plus grande différence entre les sorties du circuit et celles du modèle, sur l'ensemble de tous les stimuli pouvant être appliqués. La fonction coût dépend donc des paramètres du modèle (x) et des stimuli (i):

$$c(x, i) = \frac{1}{2} \cdot \int_0^T \| v_1(t) - v_2(t) \| dt$$

où v1 et v2 désignent les sorties correspondantes du circuit et du modèle. Précisons que les stimuli "pire-cas" choisis par *Casinovi et al.* sont représentés en Figure 18 et sont paramétrés par les points de transition, $t_{j,k}$, les valeurs M_j étant fixées arbitrairement.

Le calcul des sensibilités intervient lors de la recherche d'un minimum local de la fonction coût *c(x,i*), recherche qui nécessite la détermination du gradient de *c*.

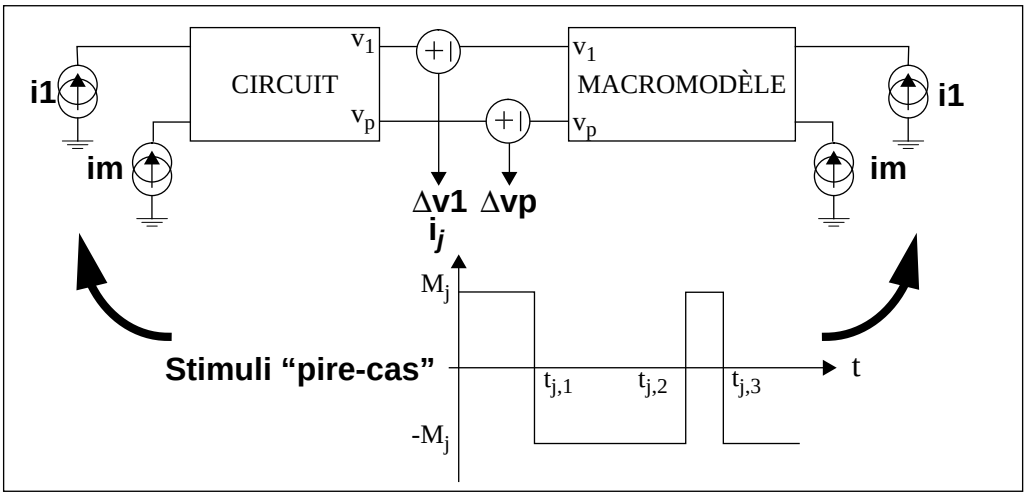


Figure 18 Stimuli pire-cas pour l'optimisation "temporelle" [CaS91]

L'algorithme décrit par *Ju et al.*[JRS91], dans le cadre du logiciel *iMAVERICK*, est basé sur des ***simulations de Monte-Carlo*** du macro-modèle. Il permet de déterminer un modèle optimal et non pas seulement "localement optimal" comme dans l'approche précédente. Cependant, pour limiter le coût de calcul numérique de cette méthode stochastique, une division de l'espace des paramètres est réalisée, chaque sous-domaine étant caractérisé par la valeur d'une fonction coût, calculée à partir des échantillons précédents et par comparaison des courbes de sortie entre le circuit et le modèle. Une fonction de probabilité est évaluée à chaque tirage aléatoire, pour déterminer si le nouvel échantillon est susceptible d'augmenter la précision de la fonction d'objectif.

D'autre part, pour la comparaison des courbes, une métrique qui tolère de faibles erreurs

de phase et qui ignore les oscillations numériques introduites par les algorithmes d'analyse temporelle, est recherchée. Deux métriques ont été ainsi utilisées pour caractériser la différence entre des courbes x et y, dans le domaine temporel ou fréquentiel:

- $d(x,y) = \|x-y\|_1$, avec $\|z\|_1 = \int_0^T |z(t)|dt$, est adaptée aux réponses fréquentielles,
- une nouvelle courbe $E_z(t) = inf_{\tau \in [0,T]} \sqrt{z^2(\tau) + s^2 \cdot (\tau - t)^2}$ est définie pour la comparaison des réponses temporelles, avec $z(t) = x(t) - y(t)$. Cela permet de réaliser un "filtrage passe-bas" de z(t). Le paramètre "s" effectue la conversion du temps en volt et permet d'ajuster le degré du filtrage. Enfin, la distance entre x et y est calculée selon l'expression: $d(x,y) = \|E_{x-y}(t)\|_1$.

7 Conclusion

Ce chapitre a tenté de mettre en évidence les points clés de l'écriture de modèles comportementaux analogiques et mixtes à l'aide soit d'un langage *structurel* de type SPICE selon la technique de macro-modélisation, dédiée à l'analogique, soit d'un langage *comportemental*, pouvant être adapté à la description de systèmes analogiques/digitaux de haut-niveau. Les principales méthodologies de définition des équations et des paramètres ont été de plus présentées.

Certaines de ces techniques de modélisation ont été mises en oeuvre au cours de cette thèse dans le cadre du développement de deux outils d'aide à l'élaboration de modèles comportementaux analogiques-digitaux:

- des ***bibliothèques fonctionnelles*** proposant des modèles génériques, paramétrables adaptés à la représentation de systèmes et de circuits,
- un ***outil interactif de caractérisation analogique***, intégré dans l'environnement de conception utilisé à SGS-Thomson. Au contraire des outils présentés dans ce chapitre, il ne nécessite pas l'apprentissage d'un langage de commande de simulation et établit un lien direct entre la caractérisation et le calcul des paramètres des macro-modèles.

Précisons enfin que l'optimisation de macro-modèles a été abordée au début de cette thèse par utilisation de l'outil *Pluto*, pour la modélisation du *slew-rate* d'un op-amp. Ce travail n'a cependant pas été poursuivi entre autres parce que Pluto est lié au simulateur électrique ST-

Spice [STS92], interne à SGS-Thomson, et qui possède des capacités limitées de macro-modélisation. D'un commun accord avec SGS-Thomson, il a été décidé d'attendre la sortie du produit *SimPilot* [Sim94], conçu à partir de *Pluto* mais associé au simulateur *Eldo* qui offre, quant à lui, un langage de modélisation comportementale HDL-A.

Chapitre 4: Bibliothèque fonctionnelle de haut-niveau

1 Introduction

Pour faciliter l'application de la méthodologie de conception *top-down* décrite au Chapitre 2, il est indispensable de fournir au concepteur une large bibliothèque de modèles comportementaux, tels que des modules mathématiques, amplificateurs, filtres, convertisseurs: ils permettent aux concepteurs d'étudier un système complet très rapidement afin d'en définir les spécifications avec précision. Ce sont des modèles qui décrivent les fonctions nominales et éventuellement un certain nombre de non-linéarités, selon les objectifs du concepteur. Ils sont indépendants de toute technologie mais sont très largement paramétrables. Ils ont été entre autres utilisés pour la description d'un système électronique de sécurité automobile concernant le déclenchement d'*air-bag*.

2 Objectifs

Cette bibliothèque, décrite par le Tableau 7, propose un ensemble de modèles analogiques, digitaux et mixtes. Trois types de modèles sont actuellement disponibles: les macro-modèles SPICE, les macro-modèles proposés par le simulateur Eldo et des modèles comportementaux. Le développement de ces derniers a constitué l'un des buts de cette thèse et a débuté par l'utilisation du langage FAS. Ces modèles FAS ont ensuite été traduits vers le langage compilé C-FAS, plus rapide, grâce au traducteur automatique FAS vers CFAS. Enfin, ils ont été tous transférés en HDL-A à l'aide du traducteur FAS vers HDL-A et en effectuant certaines modifications manuelles pour mettre à profit la puissance de modélisation digitale de ce nouveau langage.

Les objectifs qui ont présidé à l'élaboration des modèles comportementaux sont les suivants:

- permettre la description de systèmes complexes et mixtes,
- réduire les temps de simulation et les problèmes de convergence du simulateur analogique,

Tableau 9 Contenu de la bibliothèque fonctionnelle de haut-niveau

Catégorie	Type	Modèle Fonctionnel
Mathématique	**SPICE**	adder, substractor, integrator, differentiator, multiplier, divider, shift
	FAS / HDL-A	sin, cos, tan, exp, Log, abs, inverse, bounded_integrator
Amplificateur	**SPICE**	Amp_Constant, Amp_Deadzone, Amp_Limiting, Amp_V_controlled
	Eldo	opamp, opampDif
Comparateur	**Eldo**	comparator, comparatorDif
	FAS / HDL-A	Schmitt_trigger, Window_comparator, P.F.D. (Phase-Frequency-Detector)
Filtre	**Eldo**	H(s), H(z), highpass, lowpass, bandpass, bandstop, zero
	FAS / HDL-A	PID (Proportional-Integrator-Derivator) H(p,z) (Laplace transfer function with complex roots)
Générateur	**SPICE**	cycler, delay
	FAS / HDL-A	V.C.O. (Voltage Controlled Oscillator) P.W.M. (Pulse Width Modulator)
Logique	**Eldo**	inv, and2, and3, or2, or3, nand2, nand3, nor2, nor3, xor2
	FAS / HDL-A	RS-, D-, JK-, T- flip-flops Monostable multivibrators (one shot / retriggable) Astable multivibrator
Mixte	**Eldo**	dac, adc (4-16 bits)
	FAS / HDL-A	dac, adc (6 bits), Compression (6 bits), Switch, Multiplexers (resistive / capacitive), Sample&Hold (internal / external clock)
Alimentation	**HDL-A**	Step-up, Step-down, Inverting DC-DC converters, Regulator

- offrir une précision suffisante et une grande fiabilité,
- décrire la fonction nominale et certaines caractéristiques analogiques telles que les impédances d'entrée/sortie ou les temps de propagation,
- proposer des modèles paramétrables,
- détecter les erreurs de définition des paramètres de l'utilisateur et remplacer les paramètres hors-gamme par des valeurs par défaut en avertissant l'utilisateur,
- éviter les erreurs mathématiques du type division par zéro,

- introduire l'utilisation d'un langage comportemental, tel que FAS ou HDL-A, dans les méthodologies de conception. À ce titre, les modèles de la bibliothèque pourront servir de base à l'élaboration de modèles plus complexes décrivant des phénomènes spécifiques à un circuit dans une approche *bottom-u*p de raffinement. Le code est donc accessible.
- présenter un code aussi simple et clair que possible,
- fournir une documentation précisant les domaines de fonctionnement et caractéristiques des différents modèles.

Le but de ce chapitre n'est pas de présenter tous les modèles qui ont été développés mais plutôt de décrire la démarche suivie pour chaque catégorie de modèles. Afin de présenter les méthodes utilisées pour l'élaboration des modèles fonctionnels, il est intéressant de les classer suivant la manière de traiter les signaux de façon continue, quantifiée ou discrète. Le Tableau 10 présente les quatre cas extrêmes..

Tableau 10 Classification des fonctions de modélisation

	Temps continu	**Temps discrétisé**
Niveaux continus	*Domaine analogique* bloc mathématique, filtre	*Domaine échantillonné* S&H, découpage
Niveaux quantifiés	*Domaine quantifié* comparateur, convertisseur A/D	*Domaine digital* bascule, multivibrateur

Le domaine analogique se caractérise par des signaux continus, solutions d'équations différentielles et pouvant avoir à la fois une représentation temporelle et fréquentielle. Le bloc décrivant une fonction de transfert de Laplace est le modèle type de ce domaine.

Certains composants, comme les comparateurs, sont fortement non-linéaires et possèdent des niveaux de quantification précis. Seule la description temporelle est alors intéressante. Dans le cadre de l'utilisation d'un simulateur analogique, les transitions entre niveaux de quantification doivent posséder des temps de commutation non-nuls, pour des raisons de convergence. Deux types de fronts ont été envisagés: les fronts linéaires et les fronts *RC*. Des fonctions permettant de détecter des franchissement de seuils de façon précise doivent d'autre part être utilisées.

Les composants *échantillonnés*, comme le *sample & hold*, le modulateur à largeur

d'impulsions, les alimentations à découpages, sont des circuits commandés par une horloge, qui peut être externe ou interne. Des fonctions spéciales sont disponibles pour imposer au simulateur analogique des pas de temps précis. Si la description temporelle est généralement la plus intéressante, il est parfois nécessaire d'adopter une modélisation mathématique en terme de valeurs moyennes pour des analyses fréquentielles. C'est par exemple le cas des alimentations à découpages, dont la simulation AC est requise pour l'étude de la stabilité des chaînes de puissance [Kar94].

Restent enfin les composants digitaux, comme les bascules, dont les modèles sont basés sur un ensemble de relations logiques. Ils possèdent de plus des interfaces d'entrée/sortie qui permettent de détecter des évènements en entrée (conversion A/D) et de générer des signaux analogiques en sortie (conversion D/A). HDL-A est bien adapté pour développer ce genre de modèles.

Enfin, les exemples présentés sont décrits à l'aide du langage HDL-A dont les caractéristiques sont détaillées dans le cadre de la description du traducteur FAS vers HDL-A.

3 Composants analogiques

3.1 Blocs mathématiques

Ces modèles ont été développés pour compléter l'ensemble des macro-modèles mathématiques SPICE de la bibliothèque, par de nouvelles fonctions trigonométriques, logarithmiques, exponentielles... Les langages comportementaux FAS puis HDL-A ont été préférés à SPICE pour des raisons de précision et de rapidité de simulation. La Figure 19 représente la structure générale de ces modèles qui prennent aussi en compte les résistances d'entrée et de sortie.

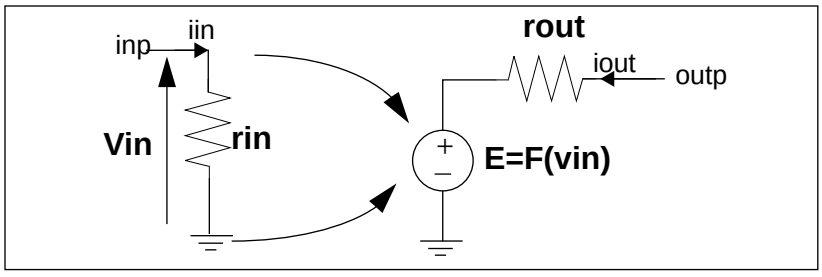


Figure 19 Structure générale des modèles mathématiques

La modélisation de ces résistances a surtout été introduite à titre d'exemple car ce type de

modèle est plutôt utilisé pour des descriptions haut-niveau, pour lesquelles les signaux représentent des variables mathématiques. Ainsi, par défaut la résistance d'entrée *rin* est très élevée et la résistance de sortie *rout* nulle. Le code HDL-A est basé sur la structure suivante:

```
ENTITY cosinus IS
   GENERIC (rin,rout : REAL) ; -- paramètres
   PIN (inp,outp : ELECTRICAL) ; -- bornes électriques
END ENTITY cosinus;

ARCHITECTURE ideal OF cosinus IS
        CONSTANT rinl,routl : REAL ; -- constantes
        STATE vin,iin,iout : ANALOG ; -- variables analogiques
BEGIN
   RELATION
        PROCEDURAL FOR INIT =>
             rin := 1.0e12;            -- résistance d'entrée
             rout := 0.0;              -- résistance de sortie
             IF (rin<1.0e-12) THEN
                  REPORT "warning: rin < 1e-12 --> rin set to 1e-12" ;
                  rinl := 1.0e-10;
             ELSE rinl := rin;
             END IF;
             IF (rout<0.0) THEN
                  REPORT "warning: rout < 0 --> rout set to 0.0" ;
                  routl := 0.0;
             ELSE routl := rout;
             END IF;
        PROCEDURAL FOR DC,TRANSIENT, AC =>
             vin := inp.V;
             inp.I %= iin; -- application du courant en entrée
             outp.I %= iout; -- application du courant en sortie
        EQUATION(iin,iout) FOR DC, TRANSIENT, AC =>
             -- définition du courant d'entrée:
             vin == rinl*iin;
             -- définition du courant de sortie:
             outp.V == F(vin)-iout*routl;
   END RELATION ;
END ARCHITECTURE ideal;
```

où `NOM` est le nom du modèle et `F` la fonction mathématique modélisée (sin, cos, ln, expo...). Dans le cas des fonctions *logarithme*, *exponentiel*, *inverse* et *tangente*, des contrôles sont effectués en plus pour éviter toute erreur mathématique. Des paramètres supplémentaires de saturation de sortie sont introduits pour déterminer les valeurs limites correspondantes de la tension d'entrée.

Le bloc "`PROCEDURAL FOR DC, TRANSIENT, AC =>`" est utilisé pour les équations analogiques explicites, comme la lecture de la tension d'entrée, et pour les équations de

connexion qui définissent la contribution en courant du modèle sur les deux bornes `inp` et `outp`. Précisons que ces courants circulent par défaut entre ces bornes et le noeud de référence du simulateur concernant le domaine électrique. Notons qu'une évolution possible serait de prendre en compte des E/S différentielles.

Le bloc "`EQUATION(iin,iout) FOR DC,TRANSIENT,AC=>`" regroupe les deux équations implicites qui modélisent les résistances d'E/S (`rinl`, `routl`) et permettent de définir les courants qui les traversent (`iin`,`iout`).

Enfin, le bloc d'initialisation "`PROCEDURAL FOR INIT =>`" permet de fixer les valeurs par défaut des paramètres. Ce bloc permet aussi de contrôler la validité des valeurs définies par l'utilisateur lors de l'appel du modèle, les valeurs définitives étant stockées dans des constantes qui sont utilisées dans les équations analogiques.

3.2 Fonction de transfert de Laplace

3.2.1 Contrôleur PID

La description de la fonction de transfert d'un contrôleur Proportionnel-Intégral-Dérivateur met en jeu les opérateurs de dérivation et d'intégration de HDL-A et se résume aux équations analogiques suivantes:

```
PROCEDURAL FOR DC =>
     vin := [inp,inm].V;
     outp.V %= kp*vin ;
PROCEDURAL FOR TRANSIENT,AC =>
     vin := [inp,inm].V;
     outp.V %= kp*vin + ki*INTEG(vin) + kd*DDT(vin);
```

où `inp`, `inm` et `outp` sont les bornes d'E/S, `INTEG` est la fonction d'intégration HDL-A qui ne peut être utilisée que pour les analyses temporelles et fréquentielles, et `DDT` est l'opérateur de dérivation. Enfin, `kp`, `ki` et `kd` désignent les trois paramètres du contrôleur.

3.2.2 Fonction de transfert à racines non-nulles

La fonction de transfert de Laplace qui a été modélisée est du type:

$$H(s) = \frac{\left(1+\frac{s}{z_1}\right)\ldots\left(1+\frac{s}{z_n}\right)}{\left(1+\frac{s}{p_1}\right)\ldots\left(1+\frac{s}{p_m}\right)} = \frac{1+a_1\cdot s+\ldots+a_n\cdot s^n}{1+b_1\cdot s+\ldots+b_m\cdot s^m},$$

de zéros $z_1 \ldots z_n$ et de pôles $p_1 \ldots p_m$ (réels ou complexes). L'équation différentielle correspondante a été décrite sous la forme d'une ***équation implicite*** d'inconnue v_{out}:

$$v_{out} + b_1 \cdot \frac{dv_{out}}{dt} + \ldots + b_m \cdot \frac{d^m v_{out}}{dt} = v_{in} + a_1 \cdot \frac{dv_{in}}{dt} + \ldots + a_n \cdot \frac{d^n v_{in}}{dt}$$

où *vin* est la tension d'entrée, la grandeur v_{out} étant appliquée sur la borne de sortie. Les dérivées successives sont déterminées de façon itérative, la dérivée d'ordre n étant égale à la dérivée de celle d'ordre n-1. Notons que pour une analyse transitoire, il est indispensable de passer par des variables analogiques intermédiaires, de type `STATE ANALOG`, qui mémorisent les valeurs des dérivées à chaque pas de temps. Le description analogique est la suivante:

```
ARCHITECTURE ideal OF TFC IS
   CONSTANT ncz, ncp: INTEGER; nb de coefficients du numérateur et dénominateur
   CONSTANT sqinv2pi : REAL; carré de l'inverse de la constante HDL-A twopi=2PI
   CONSTANT cz : real_vector (0 to max) ; -- coefficients du numérateur
   CONSTANT cp : real_vector (0 to max) ; -- coefficients du dénominateur
   VARIABLE i,j,k: INTEGER;
   VARIABLE c1, c2: REAL; -- coefficients des polynômes du 2nd degré
   STATE vin : analog_vector (0 to max); -- vecteurs des dérivées de vin
   STATE vout : analog_vector (0 to max); -- vecteurs des dérivées de vout
   STATE right,left: ANALOG ; -- membres de l'équation différentielle

   PROCEDURAL FOR DC, AC, TRANSIENT =>
        right := [inp,inn].V;
        IF nz>0 THEN -- numérateur
             vin(0) := [inp,inn].V;
             FOR i IN 1 TO ncz LOOP
                  vin(i) := DDT(vin(i-1));
                  right := right + cz(i)*vin(i);
             END LOOP;
        END IF;
        left := vout(0);
        IF np>0 THEN -- dénominateur
             FOR i IN 1 TO ncp LOOP
                  vout(i) := DDT(vout(i-1));
                  left := left + cp(i)*vout(i);
             END LOOP;
        END IF;
        outp.V %= vout(0);

   EQUATION(vout(0)) FOR DC, AC, TRANSIENT =>
        left == av*right ; -- équation différentielle
...
END RELATION ;
END ARCHITECTURE ideal;
```

où `nz` est le nombre de zéros, `np` le nombre de pôles, `ncz` est l'ordre du numérateur et `cz` le

vecteur complexe des coefficients du numérateur, `ncp` est l'ordre du dénominateur et `cp` le vecteur complexe des coefficients du dénominateur, `vin` le vecteur analogique des dérivées successives de la tension d'entrée `[inp,inn].V` et `vout` le vecteur analogique des dérivées successives de la tension de sortie. Enfin, les grandeurs `left` et `right` correspondent aux deux membres de l'équation différentielle qui a pour inconnue le premier élément de `vout`, c'est à dire `vout(0)`.

Les coefficients des numérateurs et dénominateurs sont calculés de manière itérative dans la partie d'initialisation du modèle, à partir des listes de pôles et zéros complexes définis par l'utilisateur. L'algorithme utilisé correspond au développement polynômial des facteurs du premier ordre, de la forme $1 + \frac{s}{\omega_i}$, où la pulsation est calculée par l'expression $\omega_i = 2\pi \cdot p_i$, et du second ordre $1 + c_1 \cdot s + c_2 \cdot s^2 = \left(1 + \frac{s}{\omega_i}\right) \cdot \left(1 + \frac{s}{\overline{\omega_i}}\right)$, dans le cas de pôles complexes conjugués, où les coefficients c1 et c2 sont définis par les formules: $c_2 = 1 / \left\|\omega_i\right\|^2$ et $c_1 = 2 \cdot Re(\omega_i) / \left\|\omega_i\right\|^2$. Notons que pour spécifier une paire de pôles complexes conjugués, la définition d'un seul pôle complexe par l'utilisateur est requise.

Par exemple, le vecteur des coefficients du dénominateur est défini par les instructions suivantes:

```
PROCEDURAL FOR INIT =>
   sqinv2pi := (1.0/twopi)**2;
   IF np>0 THEN
        cp(0) := 1.0;
        FOR i IN 1 TO max LOOP cp(i) := 0.0; -- initialisation
        END LOOP;
        k := 1;
        FOR i IN 1 TO np LOOP -- boucle principale
             IF pole(i).i = 0.0 THEN -- racine réelle
                  FOR j IN k DOWNTO 1 LOOP -- développement
                  cp(j) := cp(j) + cp(j-1) / (twopi * pole(i).r);
                  END LOOP;
             ELSE -- racines complexes conjuguées
                  -- calcul des coefficients du polynôme de 2nd degré
                  c2 := sqinv2pi/(pole(i).r**2 + pole(i).i**2);
                  c1 := 2.0*twopi*pole(i).r*c2;
                  k := k+1;
                  FOR j IN k DOWNTO 1 LOOP -- développement
                       IF j>1 THEN
                            cp(j) := cp(j) + cp(j-1)*c1 + cp(j-2)*c2 ;
                       ELSE
                            cp(j) := cp(j) + cp(j-1)*c1;
                       END IF;
                  END LOOP;
             END IF;
             k := k+1;
```

```
        END LOOP;
END IF;
ncp := k;
-- idem pour le numérateur ...
```

Notons l'utilisation des variables complexes: `pole(i).r` désigne la partie réelle de l'élément `pole(i)`, et `pole(i).i` désigne la partie imaginaire. Enfin, les vecteurs pole et `zero` sont déclarés dans l'entité:

```
ENTITY TF IS
   GENERIC ( av: real; nz, np, max: integer; zero: complex_vector (1 to nz+1) ;
             pole: complex_vector (1 to np+1) );
   PIN (inp,inn,outp: ELECTRICAL) ;
END ENTITY TF;
```

3.3 Interrupteur et multiplexeur

L'interrupteur (*switch*) de la bibliothèque est modélisé sous la forme d'une résistance qui est une fonction linéaire par morceaux d'une tension de contrôle. Un multiplexeur *résistif*, basé sur deux interrupteurs, et dont le fonctionnement est décrit en Figure 20, est aussi disponible.

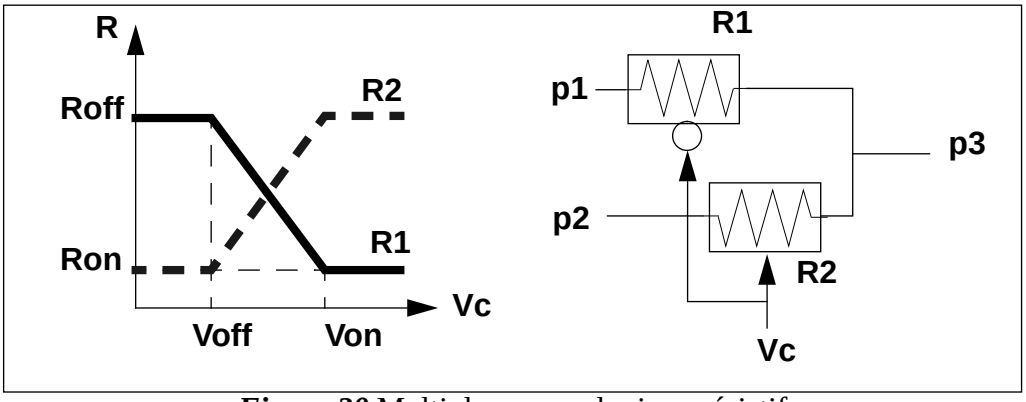


Figure 20 Multiplexeur analogique résistif

Cependant, dans le cadre de la modélisation de l'*AIR-BAG* (cf Annexe 3), un autre type de multiplexeur dit *capacitif* a été développé afin d'introduire une constante de temps de commutation. Il est basé sur l'équation différentielle de charge d'une capacité dont le terme constant correspond à l'une ou l'autre des tensions d'entrée, suivant l'état du signal de commande. Le modèle HDL-A est le suivant:

```
ENTITY mux IS
  GENERIC ( tau: real); -- constante de temps
  PORT(ts: in bit); -- signal de commande
  PIN (p1, p2, p3: electrical ); -- bornes électriques
```

```
END ENTITY mux;
ARCHITECTURE ideal OF mux IS
  STATE vout, newvout: ANALOG;
  BEGIN
    RELATION
      PROCEDURAL FOR INIT =>
             tau := 1.0e6;
      PROCEDURAL FOR DC, TRANSIENT =>
             IF (ts = '1') THEN newvout := p1.V; -- entrée p1
             ELSE newvout := p2.V; -- entrée p2
             END IF;
             p3.v %= vout; -- application sur p3
      EQUATION (vout) FOR DC, TRANSIENT =>
             tau*DDT(vout) + vout == newvout; -- charge d'une capacité
    END RELATION;
END ARCHITECTURE ideal;
```

On remarque l'utilisation d'un `PORT` digital pour transmettre l'état du signal de commande. Cet état modifie la valeur de la variable analogique `newvout` qui est soit égale à la tension de la borne p1 soit à celle de p2. Dans ce modèle, la constante de temps `tau` est indépendante du sens de charge ou de décharge. Le paragraphe 4.2 présente par contre un modèle d'échantillonneur-bloqueur pour lequel ce n'est pas le cas.

4 Comparateurs

La modélisation des comparateurs fait appel à deux notions: la détection précise de franchissement de seuil par des signaux analogiques et la génération de transitions entre les niveaux de quantification.

4.1 Détection de franchissement de seuil

Cette fonction peut être réalisée de deux manières en HDL-A:

- en accédant à la valeur calculée au pas de temps précédent, par la fonction `PREVIOUS(nom)`, où `nom` désigne une variable analogique. L'expression booléenne correspondant à la détection du franchissement par `vin` d'un seuil `vth` dans le sens ascendant s'écrit donc: `( PREVIOUS(vin)<vth AND vin>vth )`.
- par des fonctions spécialisées `RISING(vin,vth)` (sens ascendant) ou `FALLING(vin,vth)` (sens descendant), où `vin` est une grandeur analogique et `vth` la valeur du seuil. Ces fonctions permettent en particulier de synchroniser un `PROCESS` digital sur des évènements

d'origine analogique.

Pour illustrer ce dernier point, voici le code HDL-A d'un ***convertisseur A/D 1 bit***, qui comporte une partie digitale (`PROCESS`) et une partie analogique (`RELATION`):

```
ENTITY A2D_1bit IS
        GENERIC (von, voff: REAL; tpd : TIME) ;
        PORT(d : OUT BIT);
        PIN(a : ELECTRICAL) ;
END ENTITY A2D_1bit;

ARCHITECTURE ideal of A2D_1bit IS
STATE vin: ANALOG;
BEGIN
PROCESS BEGIN
        -- initialisation DC
        if vin>=von then d <= '1';
        else d <= '0';
        end if;
        LOOP -- boucle infine
        WAIT ON RISING(vin,von), FALLING(vin,voff); -- synchronisation
             -- description temporelle
             if RISING(vin,von) then d <= '1' after tpd;
             elsif FALLING(vin,voff) then d <= '0' after tpd;
             end if;
        END LOOP;
END PROCESS;
RELATION
        PROCEDURAL FOR DC, TRANSIENT =>
             vin := a.V;
        PROCEDURAL FOR INIT =>
             von := 3.0;
             voff := 1.0;
             tpd := 1ps;
END RELATION;
END ARCHITECTURE ideal;
```

Le `PROCESS` HDL-A a une structure fixe qui est basée sur une boucle infinie `LOOP`:

- Les instructions présentes avant la définition de la boucle sont exécutées lors de l'analyse du point de fonctionnement DC et permettent de trouver le point d'équilibre du modèle. Cependant, les signaux digitaux, tels que d, sont affectés au bout d'un *delta-délai* c'est à dire au premier pas de l'analyse temporelle (du moins dans la version actuelle de HDL-A). Dans une simulation mixte HDL-A, le vrai point de fonctionnement est donc le premier point transitoire.
- La seconde partie, placée après `LOOP`, concerne l'analyse temporelle. Du fait de l'instruction `WAIT ON`, l'exécution du processus est suspendue jusqu'à ce que la condition exprimée par

la liste `rising(vin,von),falling(vin,voff)` du `WAIT` soit vérifiée, c'est à dire lors d'un franchissement ascendant de `von` ou descendant de `voff`, par la tension `a.v`.

4.2 Génération de transitions

Pour assurer la continuité des signaux analogiques de sortie, plusieurs techniques peuvent être envisagées selon le niveau souhaité de précision: modélisation *exponentielle RC*, décrivant des phénomènes capacitifs réels, ou modélisation *linéaire*, de type fonctionnel.

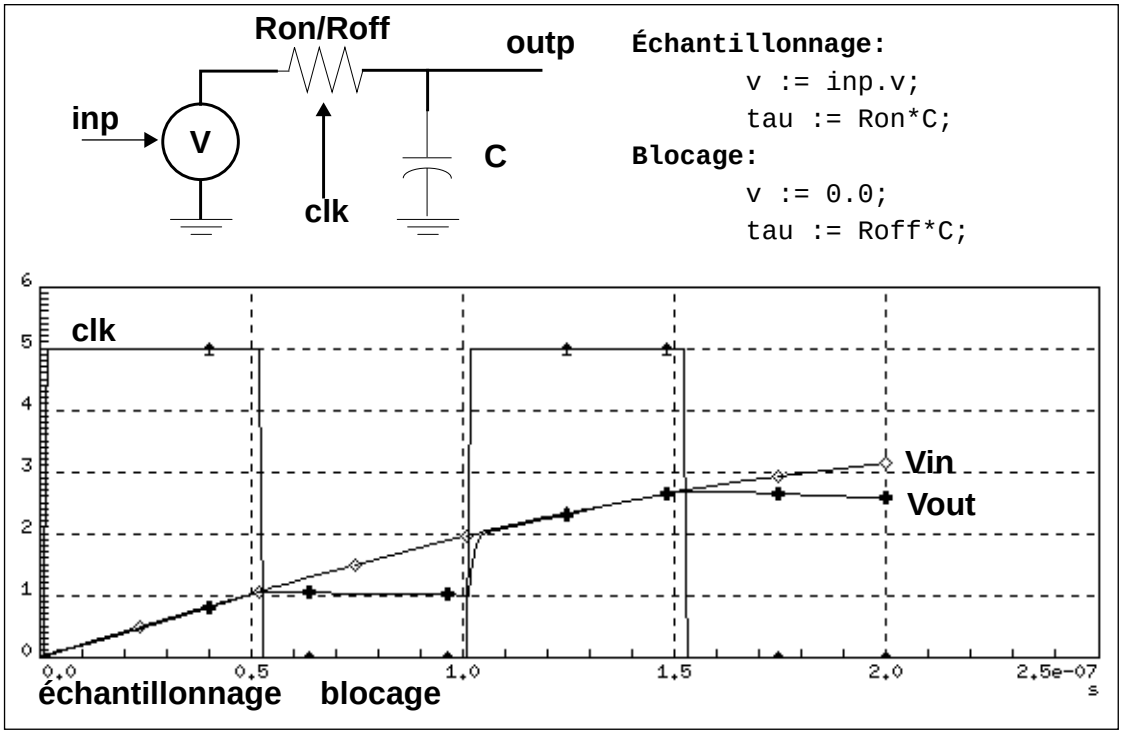


Figure 21 Modèle d'échantillonneur-bloqueur

4.2.1 Transitions exponentielles

La modélisation *exponentielle* a déjà été exposée au paragraphe 3.3 mais nous la reprenons ici sur l'exemple d'un modèle HDL-A d'échantillonnage-bloqueur pour lequel la constante de temps de l'équation différentielle est fonction du sens de charge ou de décharge de la capacité équivalente. Le schéma de principe correspond à la Figure 21. D'autre part, le modèle suivant, nommé `SH`, est décrit à l'aide d'une horloge digitale externe `clk`:

```
ENTITY SH IS
    GENERIC (stc, -- constante de temps durant l'échantillonnage
                  htc : REAL) ; -- constante de temps durant le blocage
    PORT(clk: IN BIT); -- horloge digitale
```

```
    PIN (inp,outp : ELECTRICAL) ; -- E/S analogique
END ENTITY SH;

ARCHITECTURE ideal OF SH IS
        CONSTANT stcl,htcl : REAL ; -- constantes
        STATE vout, expr : ANALOG ; -- variables analogiques
BEGIN
RELATION
        PROCEDURAL FOR DC =>
         expr := vout-inp.V;
         outp.V %= vout;
        PROCEDURAL FOR TRANSIENT =>
             IF clk = '1' THEN          -- échantillonnage
                  expr := DDT(vout)+(vout-inp.V)/stcl;
             ELSE                       -- blocage
                  expr := DDT(vout)+vout/htcl;
             END IF;
             outp.V %= vout;
        PROCEDURAL FOR INIT =>
             stc := 1.0e-6; htc := 100.0; -- constantes de temps
             IF (htc<=0.0) THEN htcl := 100.0; -- contrôles de cohérence
             ELSE htcl := htc;
             END IF;
             IF (stc<=0.0) THEN stcl := 1.0e-6;
             ELSE stcl := stc;
             END IF;

        EQUATION(vout) FOR DC, TRANSIENT => -- équation différentielle
             (expr) == 0.0;
END RELATION ;
END ARCHITECTURE ideal ;
```

Dans ce modèle, on remarque que la forme de l'équation différentielle dépend de l'état de blocage ou d'échantillonnage, défini par le niveau de l'horloge (`clk`). Comme les instructions conditionnelles ne sont pas permises dans le bloc des équations implicites `EQUATION` (du moins dans la version actuelle de HDL-A), une méthode possible est d'utiliser une variable analogique `expr` pour stocker la valeur du membre de gauche de l'équation différentielle.

4.2.2 Transitions linéaires

À un niveau d'abstraction plus élevé, les transitions sont modélisées par une fonction linéaire du temps, caractérisée soit par une pente fixe (*slew-rate*) soit par un temps de commutation entre deux niveaux donnés. Un délai avant la commutation peut être aussi pris en compte.

Une simple fonction linéaire du temps peut être définie en HDL-A à l'aide de la variable

current_time du simulateur analogique qui donne le temps courant:

```
vout := sr*analog(current_time-tinit)+vinit ;
```

où *vout* est la tension appliquée sur la borne de sortie, *sr* désigne la pente, *tinit* le temps de déclenchement, *vinit* la tension de la sortie à cet instant et enfin **analog**() est une fonction qui convertit le type temps (*time*) de la différence `(current_time-tinit)` en type analogique (*analog*).

Cependant, pour générer un front caractérisé par un délai et un temps de commutation, il est nécessaire d'imposer au simulateur des pas de temps précis en début et fin de rampe. C'est pourquoi, il est préférable d'utiliser les fonctions spécialisées de HDL-A telles que `SLOPE(outp.V,v1,td,tcom)`, où `outp` est la borne de sortie considérée, `v1` est la tension d'arrivée, `tcom` le temps de transition et `td` le délai avant commutation. Cependant, l'appel à une telle fonction doit être dûment contrôlé afin d'en éviter le déclenchement à chaque pas de temps, ce qui conduirait à un signal permanent. Une solution consiste à vérifier que le délai spécifié est écoulé avant d'activer de nouveau une fonction similaire sur la même borne.

Soit par exemple un ***comparateur à hystérésis***, nommé `trigger`, de seuils *von* et *voff*, de tensions de sortie *vhi* et *vlo*, de temps de commutation *tcom et de* délai *td*. L'entité HDL-A est la suivante:

```
ENTITY trigger IS
    GENERIC (vhi,vlo,von,voff,offset,tcom,td: REAL;);
    PIN (inp,outp : ELECTRICAL) ;
END ENTITY trigger;
```

Dans le cas où le signal d'entrée est *assez lent* par rapport aux temps de délais et de commutation du comparateur, on peut utiliser l'architecture suivante, qui n'effectue aucun contrôle sur le temps de fin de délai:

```
ARCHITECTURE cas1 OF trigger IS
        CONSTANT inverting : BOOLEAN ; -- type du trigger
        CONSTANT vonl,voffl: REAL; -- seuils
        STATE vin: ANALOG ; -- signal d'entrée
BEGIN
        RELATION
        PROCEDURAL FOR DC, TRANSIENT =>
            IF inverting THEN vin := -inp.V+offset; -- décalage
            ELSE vin := inp.V-offset;
            END IF;
        PROCEDURAL FOR DC =>
            IF (vin>vonl) THEN outp.V %= vhi;
            ELSE  outp.V %= vlo;
            END IF;
```

```
        PROCEDURAL FOR TRANSIENT =>
             IF RISING(vin,vonl) THEN -- front d'entrée montant
                  SLOPE(outp.V,vhi,td,tcom);
             ELSIF FALLING(vin,voffl) THEN -- front d'entrée descendant
                  SLOPE(outp.V,vlo,td,tcom);
             END IF;
        PROCEDURAL FOR INIT =>
             vhi:=3.3; vlo=0.0; tcom:=1.0e-6; td:=0.0;
             IF (von>voff) THEN
                  inverting := FALSE; vonl:=von; voffl:=voff;
             ELSE inverting := TRUE; vonl:=-von; voffl:=-voff;
             END IF;
        END RELATION ;
END ARCHITECTURE cas1;
```

La position relative des seuils `von` et `voff` détermine le type de comparateur, inverseur ou non-inverseur, qui est indiqué par la variable booléenne `inverting`. S'il est inverseur, une symétrie est effectuée pour se ramener à un type non-inverseur. Le franchissement des seuils est détecté par les fonctions `RISING()` et `FALLING()`.

La Figure 22 expose des résultats de simulation pour un modèle sans délai (courbe 1) et un modèle avec délai (courbe 2). Ils permettent de mettre en évidence les problèmes rencontrés lorsque le signal d'entrée franchit les seuils alors que les transitions de sortie ne sont pas encore terminées:

- Erreur 1: lorsque la valeur finale d'un front est modifiée, en raison d'un nouvel évènement, un plateau, dont la durée est liée à la précision du simulateur, est généré.
- Erreur 2: les fronts n'ont pas une pente constante. Pour avoir une pente fixe égale au *slew-rate*, le temps de commutation devrait être fonction de la tension de départ et de cible.
- Erreur 3: la spécification d'un délai plus élevé que les constantes de temps du signal d'entrée conduit à un signal de sortie plat.

Pour éviter ces problèmes, il est possible de ***filtrer les évènements*** intervenant sur l'entrée en interdisant tout nouveau déclenchement avant la fin d'une transition. Le contrôle est effectué:

- soit en lisant la valeur présente de la tension de sortie, qui doit être aux niveaux haut ou bas pour une nouvelle transition,
- soit en calculant, au moment d'un déclenchement, le temps de fin de front qui est égal à la somme du temps courant, délai et temps de commutation et en comparant ensuite le

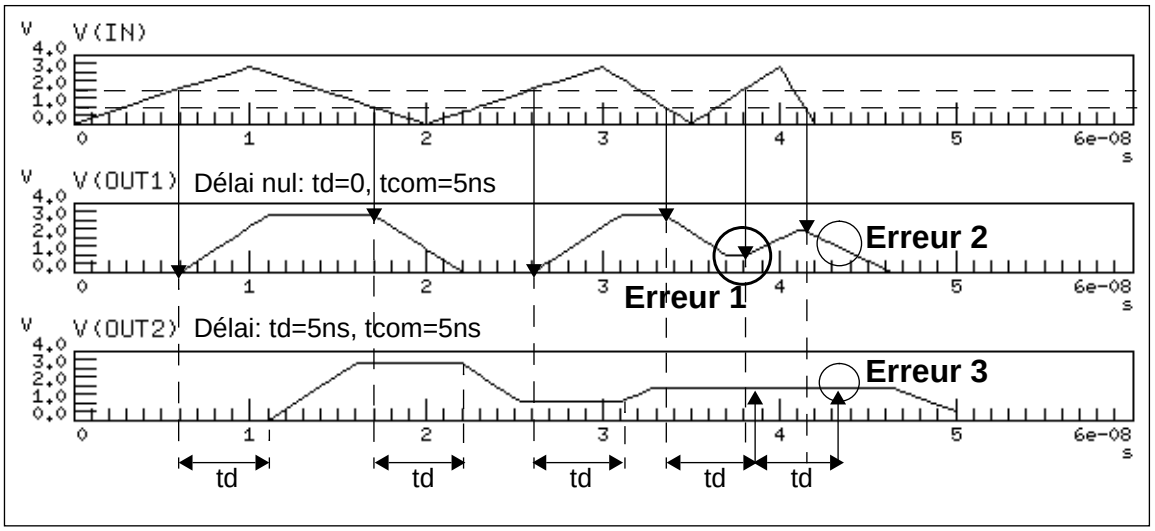


Figure 22 Simulation transitoire de comparateurs **sans contrôle digital**

temps courant à cette valeur.

Si on ne souhaite pas réaliser un tel filtrage mais effectuer un délai réel, il est nécessaire de faire appel aux possibilités de HDL-A de programmation d'évènements de signaux digitaux. Ainsi, un signal digital (`level`) qui indique l'état du comparateur est introduit, de même qu'un `PROCESS` digital. Les fonctions `RISING()` et `FALLING()` sont placées, dans ce cas, dans la liste d'évènements de l'instruction `WAIT` du `PROCESS`. Les fonctions `SLOPE()` sont ensuite commandées par le signal de contrôle `level` dont les changements d'états sont détectés par la fonction `EVENT()`. Notons que le délai est ici pris en compte dans l'affectation du signal digital, le délai spécifié dans le fonctions `SLOPE()` étant nul. De plus, le temps de commutation, `tedge`, est calculé en fonction de la valeur de sortie `vout` du pas précédent. Les résultats de l'architecture suivante sont présentés en Figure 23:

```
ARCHITECTURE cas2 OF trigger IS
        CONSTANT inverting : BOOLEAN ;
        CONSTANT slew: REAL;
        VARIABLE tedge: REAL;
        SIGNAL level: BIT; -- signal digital de contrôle
        STATE vin,vout: ANALOG ;
        ...
BEGIN
   PROCESS BEGIN -- description digitale de contrôle
         -- premier pas de temps
        IF (vin>vonl) THEN level <= '1'; ELSE level <= '0'; END IF;
        IF inverting THEN level <=  not level; END IF;
        LOOP -- boucle infinie
        WAIT ON RISING(vin,vonl),FALLING(vin,voffl); -- synchronisation
             IF RISING(vin,vonl) THEN level<='1' after tdl;
             ELSIF FALLING(vin,voffl) THEN level<='0' after tdl;
             END IF;
```

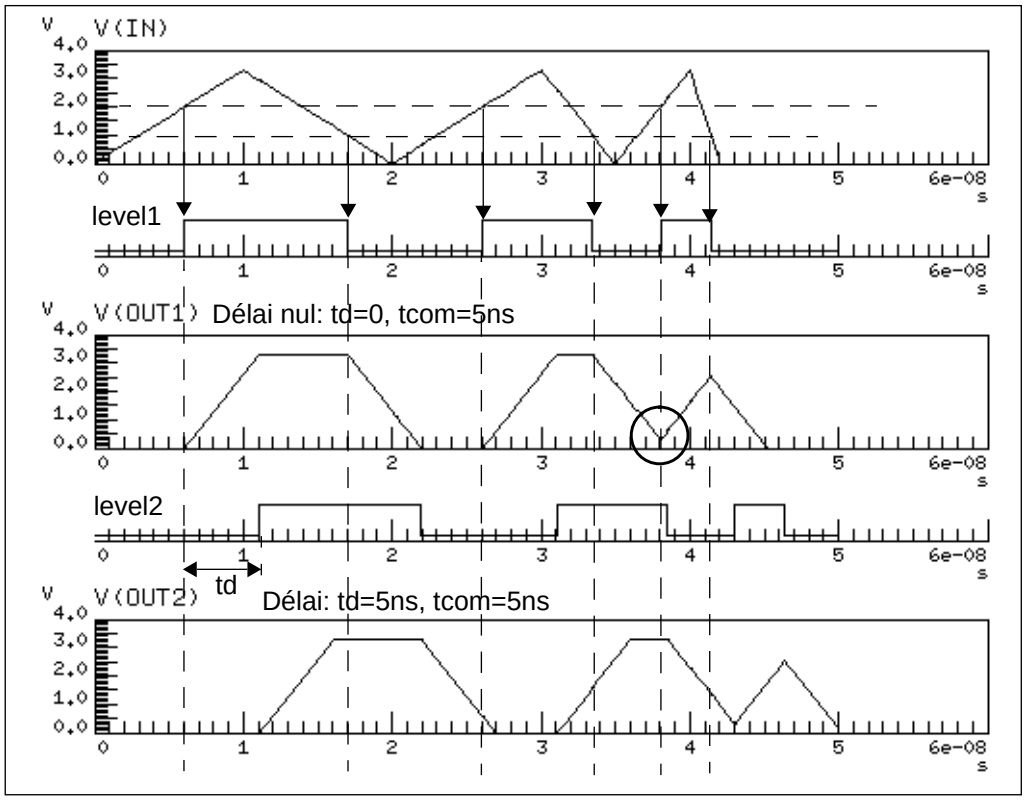


Figure 23 Simulation transitoire de comparateurs avec **contrôle digital**

```
     END LOOP;
     END PROCESS;
RELATION -- description analogique
     PROCEDURAL FOR DC, TRANSIENT =>
          IF inverting THEN vin := -inp.V+offset;
          ELSE vin := inp.V-offset;
          END IF;
     PROCEDURAL FOR DC =>
          IF level='1' THEN outp.V %= vhi; ELSE  outp.V %= vlo;
          END IF;
     PROCEDURAL FOR TRANSIENT =>
          IF EVENT(level) THEN
               vout := outp.V; -- tension de sortie de départ
               IF level='1' THEN -- transition 0->1
                    -- temps de commutation & génération du front
                    tedge:=(vhi-real(vout))/slew;
                    SLOPE(outp.V,vhi,0.0,tedge);
               ELSE -- transition 1->0
                    -- temps de commutation & génération du front
                    tedge:=(real(vout)-vlo)/slew;
                    SLOPE(outp.V,vlo,0.0,tedge);
               END IF;
          END IF;
     PROCEDURAL FOR INIT => -- initialisation
           ...
           slew:=(vhi-vlo)/tcoml; -- pente des fronts
     END RELATION ;
```

```
END ARCHITECTURE cas2;
```

On peut raffiner ce modèle en distinguant les pentes des fronts ascendants et descendants.

5 Domaine digital

Nous nous plaçons ici dans le cas de l'utilisation d'un simulateur analogique. Les modèles *digitaux*, tels que les bascules et multivibrateurs, qui ont été initialement développés à l'aide du langage analogique FAS, comportent en fait trois sections (cf Figure 24):

- un ***convertisseur A/D***, basé sur les fonctions de détection de franchissement de seuil avec hystérésis,
- un module décrivant la ***table de vérité*** à l'aide d'équations booléennes: en HDL-A, cela correspond au `PROCESS` digital,
- un ***convertisseur D/A***, basé sur les fonctions de génération de rampes.

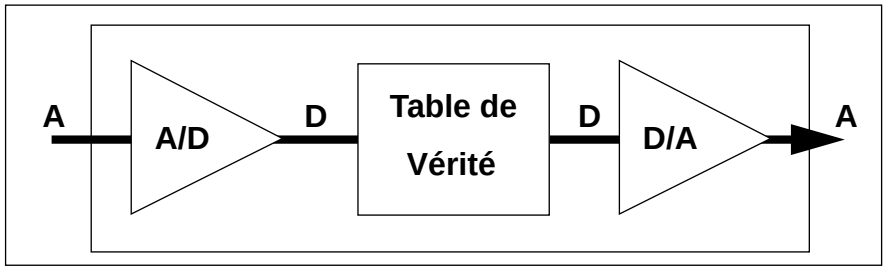


Figure 24 Fonction digitale insérée dans l'analogique

Cette structure est en fait générale à toute fonction digitale qui est connectée à des composants analogiques. Dans le cas d'un environnement de simulation mixte analogique/ digitale, une bibliothèque de convertisseurs A/D et D/A est proposée. Ces éléments fictifs doivent alors être placés par l'utilisateur aux noeuds d'interface entre la partie analogique et la partie digitale, qui est composée de modules *Verilog* ou *VHDL*.

HDL-A, qui étend VHDL au domaine analogique, permet de décrire des fonctions uniquement digitales, comme par exemple une ***bascule RS***:

```
ENTITY rsflip IS
        generic(tpd: time);
        port (r,s: in bit; q, qn: out bit);
END ENTITY ;

ARCHITECTURE digital OF rsflip IS
        variable q_on: bit;
BEGIN
   PROCESS BEGIN
```

```
        if s='0' and r='1' then q_on := '0';-- table de vérité RS
        elsif s='1' and r='0' then q_on := '1';
        elsif s='1' and r='1' then q_on := '0' ;
             report "set=1 and reset=1 are forbidden";
        else q_on := '0' ;
        end if;
        q <= q_on; qn <= not q_on;
        LOOP WAIT ON r,s;
             if s='0' and r='1' then q_on := '0';-- table de vérité RS
             elsif s='1' and r='0' then q_on := '1';
             elsif s='1' and r='1' then
             report "set=1 and reset=1 are forbidden - keep last state";
             end if;
             q <= q_on after tpd;
             qn <= not q_on after tpd;
        END LOOP;
   END PROCESS;
   RELATION
        PROCEDURAL FOR INIT =>
             tpd := 100ps;
   END RELATION;
END ARCHITECTURE ;
```

Dans ce modèle digital, la partie analogique `RELATION` ne sert qu'à l'initialisation du paramètre de délai. Dans la bibliothèque fonctionnelle, la description de cette bascule *RS* est en fait étendue au domaine analogique en réalisant des conversions A/D et D/A comme indiqué en Figure 24:

```
ENTITY rsflip IS
  generic       (von, voff, vlo, vhi: REAL; td, tcom: TIME);
  pin           (r,s,q,qn: electrical);
END ENTITY ;

ARCHITECTURE analog OF rsflip IS
        CONSTANT slew: real;
        VARIABLE tslew : real;
        SIGNAL set, reset, q_on : bit;
        STATE vr, vs, vq: analog;
BEGIN
PROCESS BEGIN
        -- conversion A/D de s.v
        IF vs>von THEN set<='1'; ELSE set<='0'; END IF;
        -- conversion A/D de r.v
        IF vr>von THEN reset<='1'; ELSE reset<='0'; END IF;
        -- table de vérité RS:
        IF set='0' AND reset='1' THEN q_on  <= '0';
        ELSIF set='1' AND reset='0' THEN q_on <= '1';
        ELSIF set='1' AND reset='1' THEN q_on  <= '0';
        REPORT "set=1 and reset=1 are forbidden";
        ELSE q_on  <= '0' ;
        END IF;
```

```
        LOOP WAIT ON set, reset, q_on, RISING(vs,von),FALLING(vs,voff),
             RISING(vr,von),FALLING(vr,voff);
             -- conversion A/D de s.v
             IF RISING(vs,von) THEN set<='1';
             ELSIF FALLING(vs,voff) THEN set<='0';
             END IF;
             -- conversion A/D de r.v
             IF RISING(vr,von) THEN reset<='1';
             ELSIF FALLING(vr,voff) THEN reset<='0';
             END IF;
             -- table de vérité RS:
             IF set='0' AND reset='1' THEN q_on  <= '0' AFTER t;
             ELSIF set='1' AND reset='0' THEN q_on <= '1' AFTER td;
             ELSIF set='1' AND reset='1' THEN
             REPORT "set=1 and reset=1 are forbidden - keep last state";
             END IF;
        END LOOP;
END PROCESS;
RELATION
        PROCEDURAL FOR INIT =>
        -- définition des valeurs par défaut
        -- pente des fronts de sortie:
        slew := real(tcom)/abs(vhi-vlo);
        PROCEDURAL FOR DC =>
             vs := s.v; vr := r.v;
             IF q_on='1' THEN vq := vhi; ELSE vq := vlo; END IF;
             q.v %= vq; qn.v %= vlo+vhi-vq;
        PROCEDURAL FOR  TRANSIENT =>
             vs := s.v; vr := r.v;
             IF EVENT(q_on) THEN -- génération des rampes: conversion D/A
                  IF q_on='1' THEN vq := vhi; ELSE vq := vlo; END IF;
                  tslew := real(abs(q.v-vq))*slew;
                  SLOPE(q.v,vq,0.0,tslew); -- front sur q
                  SLOPE(qn.v,vhi+vlo-vq,0.0,tslew); -- front sur qn
             END IF;
   END RELATION;
END ARCHITECTURE ;
```

L’entité de ce modèle mixte ne comporte que des bornes analogiques (pin). Des paramètres concernant les seuils des comparateurs d’entrée, les tensions hautes et basses, et les caractéristiques des fronts de sortie ont été de plus rajoutés. Les instructions utilisées pour les conversions A/D et D/A sont les mêmes que celles décrites aux paragraphes 4.1 et 4.2. Enfin, le signal résultant de la table de vérité, **q_on**, est utilisé pour déclencher les rampes en sortie.

6 Échantillonnage et Découpage

Sont regroupés dans cette catégorie des composants commandés par une ***horloge***

interne, tels que l'échantillonneur-bloqueur, le modulateur à largeur d'impulsions (Pulse Width Modulator) et un régulateur du rapport cyclique d'alimentations à découpage qui comporte une fonction PWM. Ces composants sont généralement inclus dans des boucles de régulation et des simulations temporelles et fréquentielles sont indispensables. La simulation ***temporelle*** d'une boucle à verrouillage de phase permet par exemple d'étudier le phénomène d'accrochage. Cependant, une telle étude peut être relativement longue en raison des hautes fréquences des signaux. Les simulations ***fréquentielles*** sont intéressantes de ce point de vue, mais elles nécessitent une description mathématique "haut-niveau" des composants.

6.1 Description temporelle

Les composants à échantillonnage et à découpage sont contrôlés par une horloge qui peut être générée soit de façon externe, par une source impulsionnelle du simulateur, soit au sein même du modèle comportemental. Dans ce dernier cas, il faut imposer au simulateur analogique des pas de temps multiples d'une période. En HDL-A, la solution consiste à faire appel aux algorithmes digitaux dirigés par évènements en définissant un signal digital périodique dont la valeur est déterminée dans la boucle infinie d'un `PROCESS`. La description analogique est alors synchronisée avec ce signal par la fonction `EVENT()`, déjà présentée dans le dernier modèle du comparateur. La partie digitale a la forme suivante:

```
PROCESS BEGIN
        CLOCK <= '0', '1' AFTER TIME(periode); -- initialisation
        LOOP WAIT ON CLOCK;
             CLOCK <= NOT CLOCK AFTER TIME(periode); -- nouvelle valeur
        END LOOP ;
END PROCESS ;
```

où `periode` correspond en fait à la demi-période du signal digital `CLOCK`, et `TIME()` est une fonction de conversion vers le type temporel *time*.

Ainsi, dans le cas d'un modulateur PWM (cf Figure 25), il est possible de générer un signal triangulaire (*vtriangle*), synchronisé avec l'horloge digitale interne `CLOCK`, et dont la valeur est ensuite comparée à la tension de la borne d'entrée (*vin*):

```
IF EVENT(clock) THEN
        tinit := current_time; -- début de la rampe
END IF;
-- définition d'une rampe
vtriangle := analog(current_time-tinit)*(vmax-vmin)/analog(periode) ;
IF (vin - vtriangle) > 0.0 THEN ... -- vout passe à l'état haut
```

```
ELSE ... -- vout passe à l'état bas
END IF;
```

Rappelons que `current_time` désigne le temps courant du simulateur analogique et `analog`() est une fonction de conversion vers le type analogique *analog*. De plus, les paramètres `vmax`, `vmin` et `periode` caractérisent la forme de la variable interne `vtriangle`. Enfin, la fonction de comparaison est simplement amorcée.

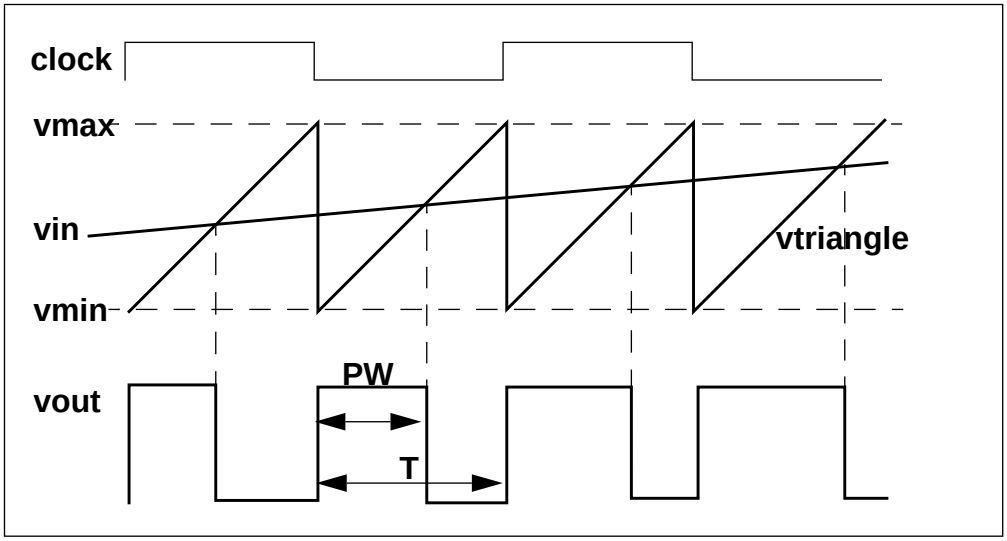


Figure 25 Signaux d'un modulateur à largeur d'impulsions P.W.M.

Une autre méthode de modélisation de la fonction PWM consiste à calculer directement la largeur des impulsions du signal de sortie à chaque évènement de l'horloge digitale `CLOCK`, en fonction de la valeur du signal d'entrée. La formule utilisée est la suivante:

$$PW = T \cdot (d_{min} + k \cdot (v_{in} - v_{min})) \text{ , avec } k = (d_{max} - d_{min}) / (v_{max} - v_{min}) ,$$

où *T* est la période d'échantillonnage, *vmax* et *vmin* sont les valeurs maximales et minimales de l'entrée, *dmin* et *dmax* sont les valeurs maximales et minimales du rapport cyclique (qui est défini par la relation $d = PW/T$). Les fronts montants puis descendants de la tension de sortie *vout* sont ensuite générés en utilisant par exemple les fonctions `SLOPE()` déjà décrites.

6.2 Description fréquentielle de convertisseurs DC-DC

Une représentation fréquentielle des composants échantillonnés est généralement nécessaire pour une étude au niveau système. C'est par exemple le cas de la description mathématique d'une *boucle à verrouillage de phase* à l'aide de la *variable de phase*, qui a été présentée au sujet de la méthodologie de conception *top-down* au Chapitre 2. De même, dans

le cas des ***alimentations à découpage*** ou ***convertisseurs DC-DC***, il est nécessaire d'effectuer des analyses fréquentielles pour étudier leur interaction avec les filtres de la chaîne de puissance [Kar94].

Un certain nombre de convertisseurs de tension, pour lesquels le découpage est effectué au secondaire du transformateur d'isolation, ont ainsi été modélisés à l'aide de HDL-A: élévateurs (*boost ou step-up*), abaisseurs (*buck ou step-down*) et inverseurs (*buck-boost ou inverting*). Signalons que ces travaux ont fait l'objet d'une publication présentée en Annexe 4. Ce document présente en particulier la modélisation temporelle d'un type de régulateur de SGS-Thomson ainsi qu'une comparaison des techniques de macro-modélisation SPICE et de modélisation comportementale HDL-A quant à la description fréquentielle.

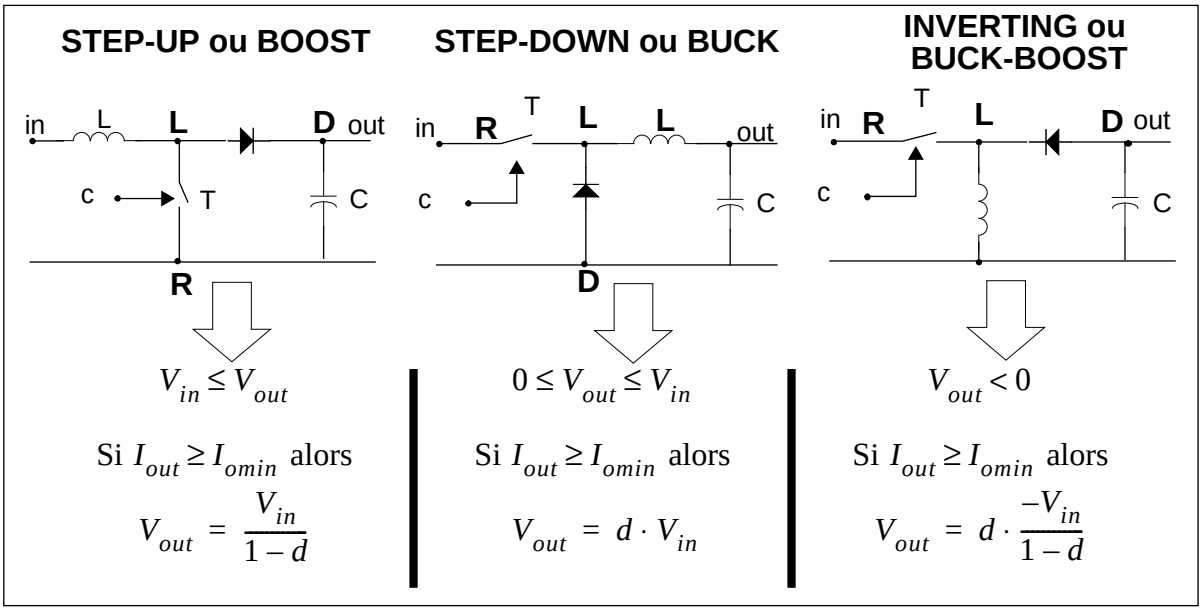


Figure 26 Les structures des trois convertisseurs DC-DC étudiés

Les alimentations qui ont été décrites se composent d'une inductance, d'une capacité, d'une diode et d'un transistor à découpage qui est commandé par une horloge définie par le rapport cyclique $d = \frac{t_{on}}{T}$, où *ton* est la largeur de l'impulsion haute pendant laquelle le transistor conduit et *T* est la période (cf Figure 26). Les formules présentées sont des relations entre ***valeurs moyennes temporelles***. Elles dépendent d'autre part du ***mode de conduction*** du convertisseur, qui est dit *discontinu* si le courant traversant l'inductance s'annule alors que le transistor est ouvert (cf Figure 27). Pour rester en conduction continue, la valeur moyenne de ce courant doit rester supérieure à la moitié de l'amplitude crête-à crête, soit $\Delta I_L / 2$. Or, la valeur moyenne de l'inductance est directement liée à la valeur moyenne du courant de sortie. Le courant de sortie doit donc être supérieur à un certain seuil I_{omin} pour rester en conduction

continue.

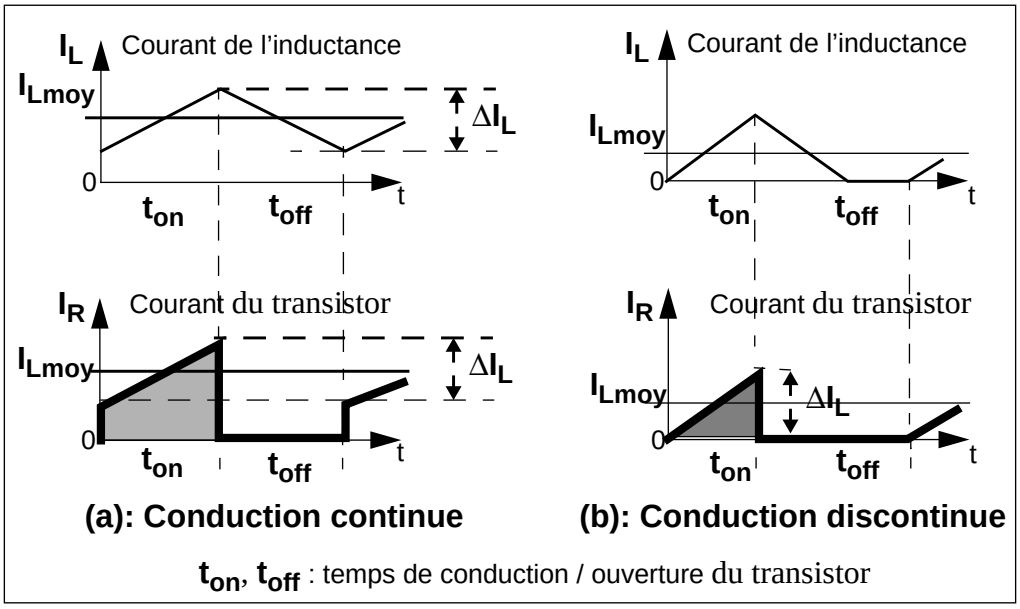


Figure 27 Définition du mode de conduction

La méthode de ***modélisation fréquentielle par valeurs moyennes*** consiste à remplacer les composants de commutation (diode et transistor) par des sources contrôlées équivalentes [Fer84], [Vop90], [Cal91]. La Figure 28 présente le modèle équivalent qui a été utilisé pour les trois types de convertisseurs: le noeud L est toujours connecté à l'inductance alors que les noeuds D et R sont connectés soit à la sortie, soit à l'entrée, soit au noeud de référence, selon le type de convertisseur (cf Figure 26). De plus, *m* désigne un coefficient de transformation qui est égal au rapport cyclique en conduction continue. Le schéma équivalent est en fait un *transformateur idéal* sans perte de puissance, caractérisé par la relation $V_{in} \cdot I_{in} = V_{out} \cdot I_{out}$. Cependant, pour une modélisation plus précise, il faudrait prendre en compte les résistances parasites des divers éléments.

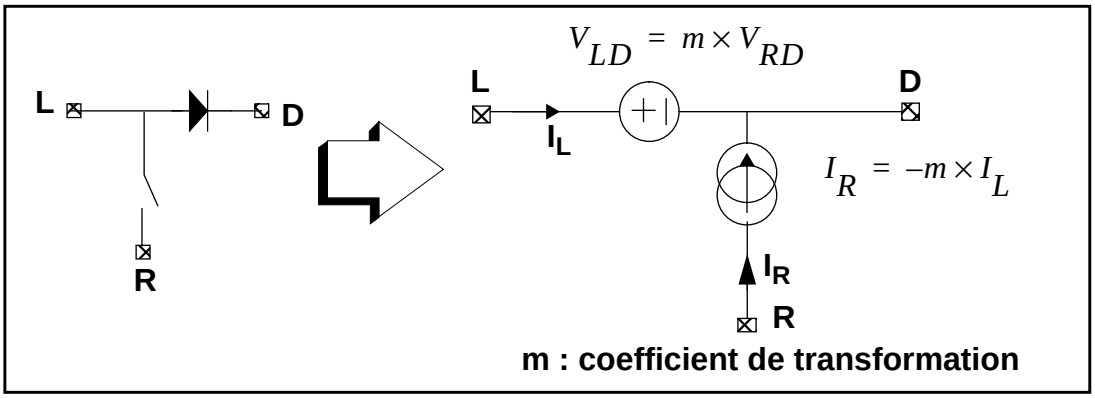


Figure 28 Modèle équivalent idéal des éléments commutés

Le modèle HDL-A, basé sur ce schéma équivalent, comporte deux équations implicites d'inconnues le coefficient *m* et le courant moyen I_L: $V_{LD} = m \cdot V_{RD}$ et $I_R = -m \cdot I_L$ (les tensions et courants sont des valeurs moyennes). La difficulté de cette modélisation consiste en fait en la définition du mode de conduction et donc de la valeur du coefficient *m*. Pour cela, la valeur moyenne du courant traversant le transistor est facilement calculée [Cal91]. D'après la Figure 27, ce courant est égal au courant de l'inductance pendant t_{on} et est toujours nul pendant t_{off}.

- En ***conduction continue***, on peut calculer la valeur moyenne de I_R à partir de l'aire grisée de la Figure 27a:
  $$I_R = \frac{|I_L| \cdot t_{on}}{T} = d \cdot |I_L|$$, où *d* est le rapport cyclique de l'horloge.

- En ***conduction discontinue***, la valeur moyenne de I_R est égal à $I_R = \frac{1}{T} \cdot \frac{\Delta I_L \cdot t_{on}}{2}$ (cf Figure 27b).
  Or, pendant le temps t_{on}, le courant de l'inductance augmente de $\Delta I_L = \frac{|v_L| \cdot t_{on}}{L}$, où v_L est la d.d.p. à ses bornes lorsque le transistor est fermé. La valeur moyenne de cette tension étant égale à la tension moyenne V_{LR} aux bornes du transistor (au signe près), on obtient par conséquent en valeur absolue:
  $$I_R = \frac{1}{T} \cdot \frac{|V_{LR}| \cdot t_{on}^2}{2 \cdot L} = \frac{T}{2 \cdot L} \cdot |V_{LR}| \cdot d^2$$

La valeur effective est en fait la plus grande des deux [Cal91], comme on peut le remarquer sur la Figure 27. Il suffit donc de calculer les deux expressions précédentes et les comparer. Cette fonction nécessite un schéma basé sur des diodes pour un macro-modèle SPICE alors qu'une simple instruction conditionnelle est utilisée en HDL-A. La structure du macro-modèle SPICE et le code HDL-A sont exposés en Annexe 4. On y trouvera aussi les résultats de simulation fréquentielle, qui sont conformes aux formules théoriques des fonctions de transfert décrites par *Ferrieux* [Fer84].

7 Application et validation: exemple de l'Air-Bag

Les modèles de la bibliothèque fonctionnelle de haut-niveau ont été validés dans des schémas de test tels qu'une boucle à verrouillage de phase, un convertisseur tension-fréquence, un extracteur de valeur moyenne R.M.S. (Root-Mean-Square) [TiS91], des compteurs et des registres à décalage, ainsi que pour la description à plusieurs niveaux hiérarchiques d'un

dispositif électronique de déclenchement d'*AIR-BAG* (cf Figure 29). Cette dernière application, proposée par la société BOSCH dans le cadre d'un projet européen JESSI, a fait l'objet d'une autre publication présentée en Annexe 3.

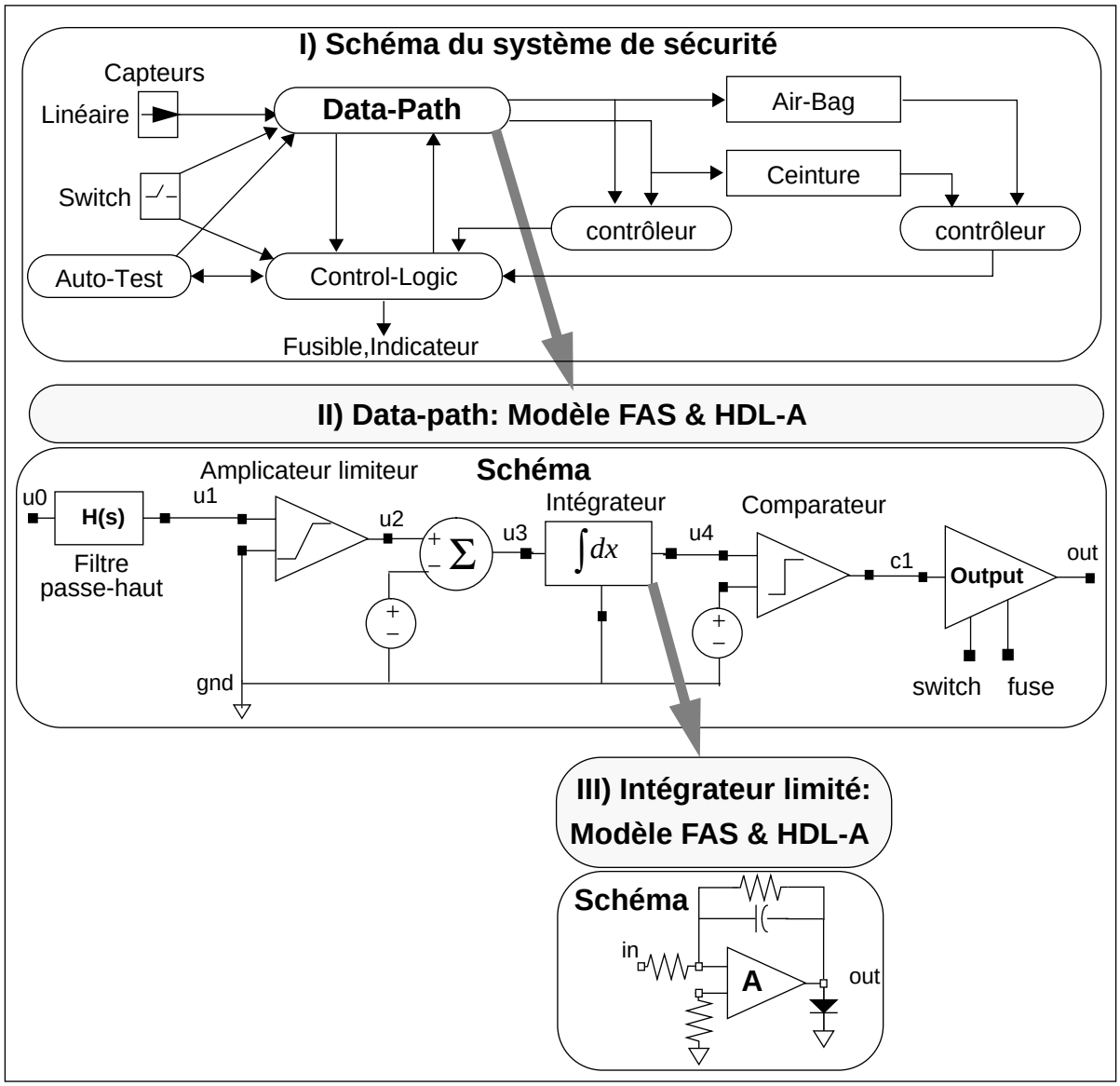


Figure 29 Les niveaux hiérarchiques du dispositif de déclenchement d'Air-Bag

Ce document présente en détail comment ce système complexe analogique/digital a pu être modélisé à l'aide d'un langage comportemental selon une approche descendante *top-down*. Des modèles comportementaux de haut-niveau ont tout d'abord été développés en FAS puis en HDL-A pour la simulation du système composé de deux capteurs d'accélération, des actionneurs qui déclenchent l'air-bag ou tendent la ceinture de sécurité, d'un module de traitement de l'accélération nommé *Data-Path*, d'un bloc purement digital de contrôle désigné *Control-Logic*, d'un module générant des signaux analogiques d'auto-test et enfin des

contrôleurs mesurant les niveaux de sortie. La description HDL-A du *Data-Path* est par exemple la suivante:

```
ENTITY airbag_datapath IS
   GENERIC(fc,a,umax,umin,offset,von,voff,vhigh,vsmall: real; tcom,td:time);
   PORT(fuse,mercury_switch: in bit);
   PIN(acceleration,out1: electrical); -- une seule sortie est ici décrite
END airbag_datapath;

ARCHITECTURE a1 OF airbag_datapath IS
   VARIABLE c1: BIT;
   SIGNAL high1,small1: BIT;
   STATE u0,u1,u2,u3,u4,u5,vnext: ANALOG;
BEGIN
PROCESS BEGIN -- DESCRIPTION DU COMPARATEUR & BLOC DE SORTIE
        IF (u4 <= uth) THEN c1 := '1'; ELSE c1 := '0'; END IF;
        high1  <= c1 and fuse and mercury_switch;
        small1 <= c1 and (not fuse or not mercury_switch);
   LOOP -- BOUCLE INFINIE
   WAIT ON fuse, mercury_switch, RISING(u4,voff),FALLING(u4,von);
                                    -- SYNCHRONISATION A/D
        IF FALLING(u4,von) THEN c1 := '1';
        ELSIF  RISING(u4,voff) THEN c1 := '0';
        END IF;
        high <= c1 AND fuse AND mercury_switch AFTER td;
        small <= c1 AND (NOT fuse OR NOT mercury_switch) AFTER td;
   END LOOP;
END PROCESS;
RELATION
   PROCEDURAL FOR INIT =>
        fc := 1.0; a := 1.0; umax := 100.0 ; umin := -35.0;
        offset := -4.0; von:= -0.2; voff:= -0.19;
        vhigh := 10.0; vsmall := 5.0; tcom := 1us ; td := 1us ;
   PROCEDURAL FOR DC, TRANSIENT =>
        u0 := acceleration.V; -- LECTURE DE L'ACCÉLÉRATION
        u2 := a*u1; -- AMPLIFICATION
        IF (u2 > umax) THEN u2 := umax;-- LIMITATION
        ELSIF (u2 < umin) THEN u2 := umin;
        END IF;
        u3 := u2 - offset; -- DÉCALAGE
   PROCEDURAL FOR DC =>
        IF (u3 > 0.0) THEN u4 := 0.0; ELSE u4 := u3; END IF;
        IF (high = '1') THEN out1.V %= vhigh;
        ELSIF (small = '1') THEN out1.V %= vsmall;
        ELSE out1.V %= 0.0;
        END IF;
  PROCEDURAL FOR TRANSIENT =>
        u5 := PREVIOUS(u4) + (u3 + PREVIOUS(u3))*TIME_STEP/2.0; -- INTÉGRATION
        IF (u5 > 0.0) THEN u4 := 0.0; ELSE u4 := u5; END IF; -- LIMITATION < 0
        IF EVENT(high) OR EVENT(small) THEN -- CONVERSION D/A
             IF (high = '1') THEN vnext := vhigh;
             ELSIF (small = '1') THEN vnext := vsmall;
```

```
            ELSE vnext := 0.0;
            END IF;
            SLOPE(out1.V,vnext,0.0,tcom); -- GÉNÉRATION DU FRONT DE SORTIE
        END IF;
  EQUATION (u1) FOR DC, TRANSIENT => -- FILTRE PASSE-HAUT
        DDT(u1) + TWOPI*fc*u1 == DDT(u0);
END RELATION
END ARCHITECTURE a1;
```

On se reportera à la Figure 29 pour la définition des différents signaux. Notons seulement que le bloc d'intégration limitée au domaine négatif n'a pas été décrit à l'aide de la fonction HDL-A nommée `integ()` mais plutôt en utilisant l'algorithme d'intégration par trapèze qui permet, à chaque pas de temps du simulateur, de contrôler le signe de l'intégrale et de stopper l'opération si elle tend à devenir positive.

Enfin, tous les blocs du système d'air-bag possèdent aussi une description structurelle, à part le module d'auto-test. Les modules à dominante analogique (capteurs, actionneurs, *Data-Path* et contrôleurs) sont décrits sous forme de schémas-blocs basés sur des éléments de la bibliothèque fonctionnelle ainsi que sur de nouveaux modèles comportementaux (intégrateur limité au domaine négatif et étages de sortie du *Data-Path*). Quant au bloc numérique *Control-Logic*, il a été décrit en utilisant des portes logiques, qui peuvent être soit des macro-modèles du simulateur Eldo, soit des modules Verilog en vue de l'utilisation du système de simulation mixte *Verilog-Eldo.* Enfin, l'intégrateur a été représenté au niveau inférieur à l'aide d'un amplificateur opérationnel et de composants analogiques élémentaires.

8 Conclusion

Cette bibliothèque de modèles fonctionnels n'est pas exhaustive mais elle comporte les principaux types de modèles comportementaux analogiques, mixtes et digitaux, dont les concepteurs pourront s'inspirer pour leurs propres applications. Différentes méthodes de modélisation ont été de plus proposées en HDL-A. Notons enfin la plupart des modèles sont aussi disponibles en FAS et C-FAS.

Chapitre 5: Bibliothèque fonctionnelle pour la modélisation d'op-amps

1 Introduction

Cette bibliothèque propose des modèles de blocs fonctionnels SPICE et HDL-A plus particulièrement adaptés à la réalisation modulaire de macro-modèles d'op-amps. Ils ont été utilisés pour décrire fidèlement le comportement d'un nouvel amplificateur de SGS-Thomson, large bande et de précision élevée, de référence MC33272. Le Tableau 11 donne la liste des modèles qui ont été ainsi validés.

Tableau 11 Modèles d'étages pour macro-modélisation d'Op-Amps

Étage	Type	Langage	Référence
Input	entrée sans transistors	SPICE, HDL-A	[CoC92]
Input_q	entrée avec transistors (npn) + slew-rate	SPICE	[AlB90] [BCP74]
Gain	gain Avd + limitation de tension + point milieu [vdd,vss]	SPICE	[AlB90]
Gain_p	gains Avd & Avcm + limitation de tension + pôle + point milieu [vdd,vss]	SPICE	[AlB90]
Gain_sr	gains Avd & Avcm + slew-rate + pôle + limitation de tension	SPICE HDL-A	[CoC92] ([ChL75])
Pole	pôle	SPICE, HDL-A	[CoC92]
Zero	zéro	SPICE, HDL-A	[CoC92]
Pole_sym	pôle + point milieu [vdd,vss]	SPICE	[AlB90]
Zero_sym	zéro + point milieu [vdd,vss]	SPICE	[AlB90]
H(s)	Fonction de transfert de Laplace	HDL-A	cf Chapitre 4
Output	résistance & capacité de sortie + limitation de courant	SPICE, HDL-A	[CoC92] + [AlB90]

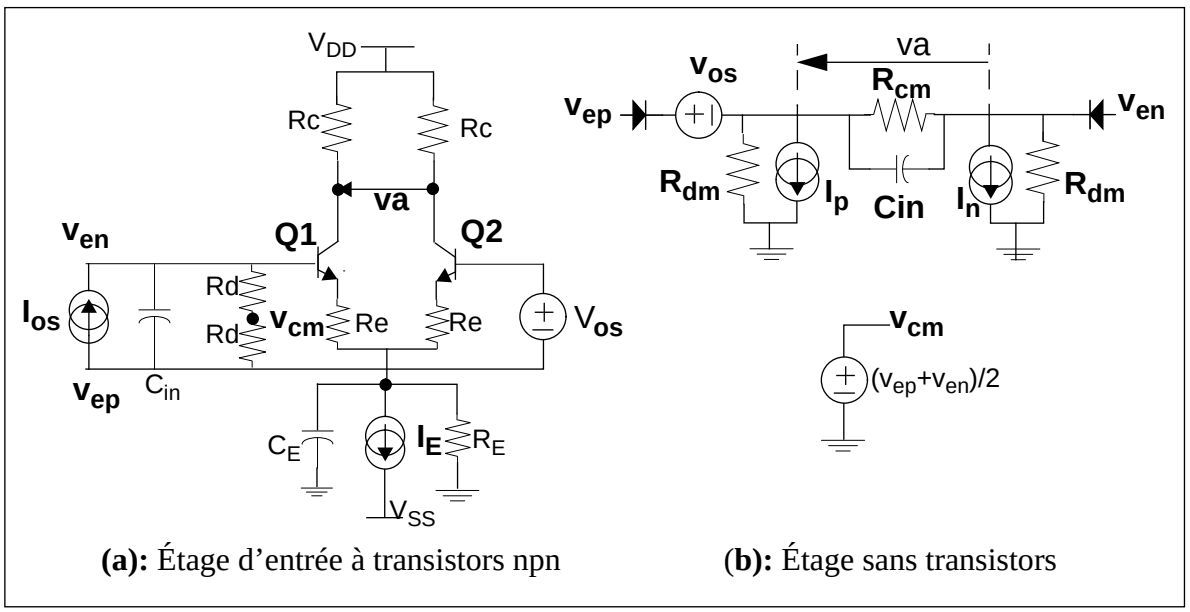
Les macro-modèles SPICE ont été élaborés à partir des structures décrites par *Boyle et al.* [BCP74], *Alexander et al.* [AlB90] et *Connelly et al.* [CoC92], et sont tout d'abord présentés au paragraphe 2. Quant aux modèles comportementaux HDL-A, ils ont été développés par simplification de certains étages SPICE et sont présentés au paragraphe 3.

2 Macro-modèles SPICE

Le code des macro-modèles SPICE est généré par un programme C qui réalise en premier lieu le calcul des valeurs des composants internes, à partir d'une liste de paramètres spécifiés par l'utilisateur. Les paragraphes suivants présentent le principe de fonctionnement des principaux étages disponibles.

2.1 Blocs d'entrée

Le but de l'étage d'entrée d'un op-amp est de décrire l'ensemble des parasites d'entrée: tensions et courants de décalage, courants de polarisation des deux bornes d'entrée, résistance et capacité. Deux signaux de sortie sont générés: le signal *différentiel*, proportionnel à la différence des tensions d'entrée $v_a = v_{ep} - v_{en}$ et le signal de *mode commun*, défini comme la valeur moyenne $v_{cm} = (v_{ep} + v_{en})/2$. La bibliothèque offre deux types de macro-modèles selon l'utilisation ou non de transistors, leurs schémas étant représentés en Figure 30.



(a): Étage d'entrée à transistors npn (**b):** Étage sans transistors

Figure 30 Les deux types d'étages d'entrée SPICE de la bibliothèque

2.1.1 Étage à transistors

L'étage nommé "*Input_q*" est basé sur une paire différentielle de transistors bipolaires npn idéaux, de gains β égaux et correspond à la structure décrite par *Alexander et al.* [AlB90]. Celle-ci présente l'avantage, par rapport à la proposition de *Boyle et al.* [BCP74], d'autoriser l'utilisation de transistors pnp ou JFET afin que le comportement du macro-modèle soit plus proche de celui du circuit. Notons que cette possibilité n'est pas encore disponible mais est envisagée.

Les tensions et courants de décalage sont décrits par des sources idéales, respectivement Vos et Ios, la valeur de Ios étant déterminée à partir des courants d'entrée de l'opamp, Ibp et Ibn par la relation: $Ios = (Ip - In)/2$. Ces courants définissent aussi le gain des transistors sachant que le courant de polarisation I_E, paramètre du modèle, est égal à $I_E = \beta \cdot (Ip + In)$. L'impédance d'entrée est prise en compte par la capacité C_{in} et le pont diviseur de résistances Rd, qui permet aussi de calculer la tension de mode commun Vcm. La résistance d'entrée du macro-modèle correspond ainsi à $2 \cdot R_d$ en parallèle avec $2 \cdot \beta \cdot (R_e + r_d)$, où rd est la résistance différentielle des transistors npn définie par $r_d = V_T/I_c = 2 \cdot V_T/I_E$, avec $V_T \approx 26mV$. Signalons que le gain de cet étage d'entrée, $R_c/(R_e + r_d)$, est fixé à 1, ce qui impose la valeur des résistances d'émetteur $R_e = R_c - r_d$, valeur qui doit être positive ou nulle.

Une contrainte est donc imposée sur la résistance de collecteur Rc: $R_c \geq r_d$. Sa valeur est d'autre part liée au *slew-rate* du macro-modèle, encore appelé *vitesse de balayage limite en sortie* et défini par la relation $S_r = max\left|\frac{dv_s}{dt}\right|$, où *vs* est la tension de sortie de l'op-amp. Cette limitation est due à une saturation du courant pouvant charger les capacités du circuit et est mesurée en configuration pire-cas suiveur. Or un tel étage différentiel limite la dynamique du signal différentiel car le courant pouvant circuler dans les résistances de collecteur Rc est au maximum égal à la valeur de I_E, lorsqu'un des transistors conduit et l'autre est encore bloqué. Dans un op-amp réel, cet étage d'entrée débiterait directement sur un bloc amplificateur, de capacité équivalente *C* et la valeur maximale de la dérivée du signal différentiel serait donc $max\left|\frac{dv_a}{dt}\right| = \frac{I_E}{C}$. Dans le cadre de la macro-modélisation modulaire, la grandeur transmise à l'étage d'amplification suivant, généralement associé au pôle dominant, est la tension différentielle *va*, dont la variation est limitée en valeur absolue à $v_{amax} = R_c \cdot I_E$. Nous considérerons pour l'instant que cette valeur limite est un paramètre du modèle qui permet de déterminer Rc, avec la condition $v_{amax} \geq r_d \cdot I_E$ soit $v_{amax} \geq 2 \cdot V_T$. La relation entre v_{amax} et

le slew-rate est explicitée au paragraphe 2.2.

De plus, en raison des dissymétries des étages internes, on peut observer que le slew-rate positif (*Srp*) correspondant à un front montant est différent du slew-rate négatif (*Srn*) défini pour un front descendant (cf Figure 31). Dans le cas d'un étage bipolaire npn, on a par exemple $S_{rp} > S_{rn}$ et le contraire pour un pnp. Pour modéliser ce phénomène, la technique de *Boyle et al.* a été reprise. Une capacité supplémentaire C_E est montée en parallèle avec la source de polarisation I_E. Au courant I_E, se superpose ainsi le courant $i_{CE} = C_E \cdot \frac{dv_{cm}}{dt}$ qui dépend du sens de v_{ep} dans le cas du montage suiveur de la Figure 31. Lors d'un front positif, Q1 se bloque et Q2 conduit donc C_E se charge instantanément, provoquant un surplus de courant qui correspond au début très raide du front de sortie v_s (cf Figure 31). Par contre, lors d'un front négatif, Q2 se bloque et Q1 conduit établissant ainsi l'égalité entre les dérivées de v_s et de v_{cm}, ce qui conduit à la relation $S_{rn} = S_{rp} / (1 + K \cdot R_c \cdot C_E)$ avec $K = 2 \cdot \pi \cdot f_p \cdot A_{vd}$, coefficient déterminé à partir du pôle dominant et du gain différentiel.

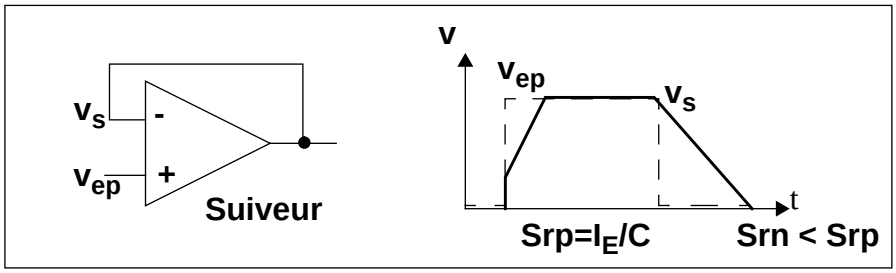


Figure 31 Dissymétrie du slew-rate

2.1.2 Étage sans transistor

L'étage nommé "Input", basé sur le schéma de *Connelly et al.* [CoC92] (Figure 30b), ne comporte par contre aucun transistor. On notera l'utilisation des sources de courants Ip et In pour décrire les caractéristiques des courants d'entrée avec $Ip = Ib + \frac{Ios}{2}$ et $In = Ib - \frac{Ios}{2}$ et la source de tension vos pour le décalage en tension. Deux diodes sont éventuellement utilisées pour remplacer des transistors bipolaires. Le sens des diodes et des deux sources de polarisation Ip et In dépendent du type de transistors: la Figure 30b correspond par exemple à un étage npn. Précisons que cet étage a par ailleurs été utilisé sans diode pour la macro-modélisation du MC33272. Sa description SPICE est la suivante:

```
.subckt Inopamp inp inm dm cm
r1 3 0   2.000e+09
r2 3 inm   2.000e+06
```

```
r3 inm 0   2.000e+09
c3 3 inm   1.400e-12
i1 3 0   1.011e-07
i2 inm 0   8.106e-08
vc inp 3   5.646e-08
ed dm 0 3 inm 1.0
rd dm 0   1.000e+06
ec cm 0 poly(2) 3 0 inm 0 0.0 0.5 0.5
rc cm 0   1.000e+06
.ends Inopamp
```

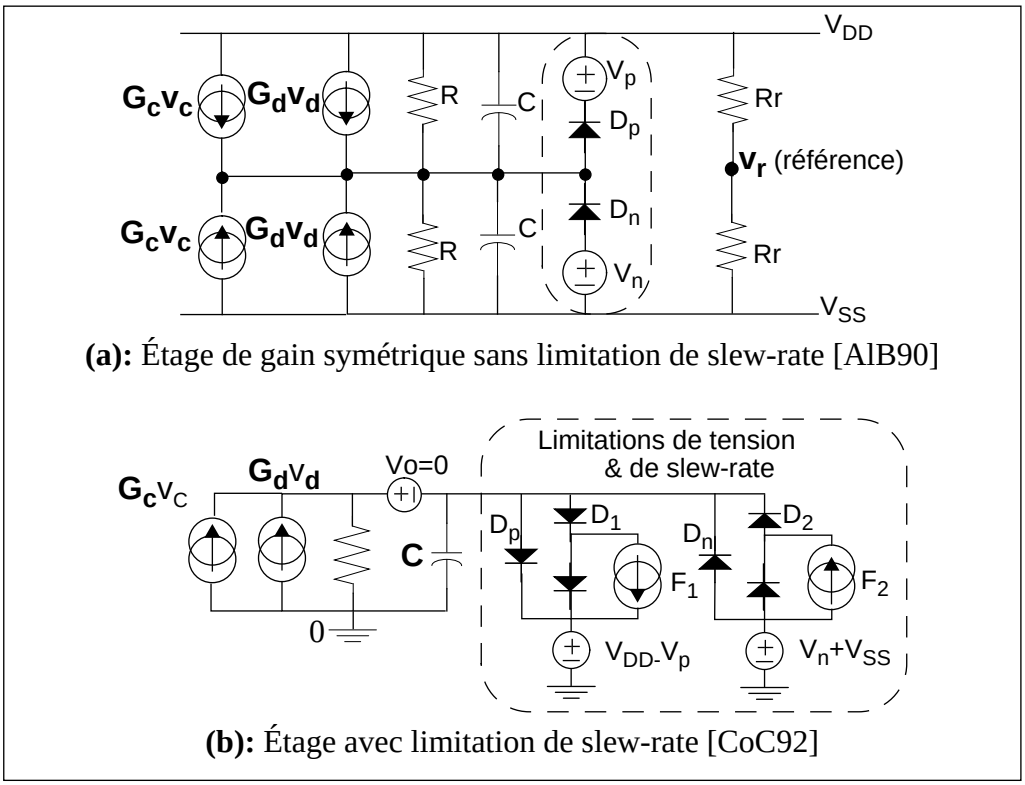
2.2 Blocs pour la fonction de transfert

Pour décrire les fonctions de transfert en mode différentiel et en mode commun, des blocs de gain (“Gain” et “Gain_sr”) et de racines (“pole”, “pole_sym”, “zero”, “zero_sym”) peuvent être utilisés en cascade. Les blocs de gain modélisent à la fois le gain différentiel *Avd*, le gain en mode commun *Avcm*, déterminé à partir du *facteur de réjection de mode commun* $CMRR = \frac{Avd}{Avcm}$, le pôle dominant de l’op-amp et enfin la limitation de tension DC. Par contre, les blocs de pôle et zéro sont de gain unitaire et sont associés aux racines secondaires.

Précisons que pour chaque bloc, deux structures sont disponibles: l’une est symétrique par rapport au point milieu des deux rails d’alimentation [AlB90] (cf Figure 32a) alors que l’autre a pour point de référence électrique le noeud 0 du simulateur [CoC92] (cf Figure 32b), l’avantage du schéma symétrique étant de modéliser le partage de courant entre les alimentations et de contrôler ainsi la consommation du macro-modèle.

Nous nous limiterons ici à la description des deux blocs de gain “Gain_p” et “Gain_sr” de la bibliothèque, représentés en Figure 32 et qui sont les plus complexes. Ces deux blocs modélisent à la fois le gain différentiel et de mode commun par les sources de courant contrôlées respectives $G_d v_d$ et $G_c v_c$, sachant que $G_d = A_{vd}/R$ et $G_c = A_{vcm}/R$, la résistance R étant arbitrairement fixée à 1kOhms.

L’association du gain et de la limitation de tension est indispensable pour éviter de générer des tensions physiquement aberrantes qui conduisent à un comportement temporel erroné en montage suiveur. Cette opération de limitation est réalisée par deux diodes idéales D_p et D_n, de coefficient d’idéalité N=0.001 et associées aux sources de tensions V_p et V_n qui représentent l’écart (en valeur absolue) entre la tension d’alimentation (resp. V_{DD} et V_{SS}) et la tension de saturation (resp. haute et basse). D’autre part, le pôle dominant (f_p) y est aussi modélisé par la capacité C dont la valeur est définie par $C = 1/(2 \cdot \pi \cdot R \cdot f_p)$. La description



(a): Étage de gain symétrique sans limitation de slew-rate [AlB90]

(b): Étage avec limitation de slew-rate [CoC92]

Figure 32 Les deux types d'étages SPICE de gain

du slew-rate est ainsi plus réaliste.

À ce sujet, le schéma de *Connelly et al.* (bloc "Gain_sr") réalise une limitation du courant de charge de cette capacité afin de limiter la dérivée du signal amplifié (cf Figure 32b). Le principe de cette limitation repose sur la mesure du courant par l'intermédiaire de la source de tension nulle Vo et sur la mise en conduction de la diode D_1, resp. D_2, lorsque le courant généré par la source contrôlée de valeur $F_1 = I(Vo) - C \cdot S_{rp}$, resp. $F_2 = -I(Vo) - C \cdot S_{rm}$, devient positif. Ainsi, le surplus de courant est dérivé dans la branche de D_1, resp. D_2, et *I(vo)* reste limité à $C \cdot S_{rp}$, resp. $-C \cdot S_{rn}$. L'effet de seuil des diodes est atténué en prenant un coefficient d'idéalité très faible (N=0.001). Dans le cas de la macro-modélisation du circuit MC33272, le code généré est le suivant:

```
.subckt Gain_sr dm cm vdd vss out
g1 0 v0 dm 0 7.328e+01
g2 0 v0 cm 0 7.476e-05
r1 v0 0   1.000e+03
v0 v0 out dc 0
c1 out 0 3.737e-07
f1 1 vmax poly(1) v0 -7.109 1.0
f2 vmin 2 poly(1) v0 -4.485 -1.0
d1 out 1 switch
d10 1 vmax switch
```

```
d2 2 out switch
d20 vmin 2 switch
dp out vmax switch
dn vmin out switch
vp vdd vmax 0.334668
vm vss vmin -0.183883
.model switch d n=0.1
.ends Gain_sr
```

Par contre, le schéma symétrique d'*Alexander et al.* (bloc "Gain") n'inclut pas cette limitation du courant de charge. Cependant, le slew-rate peut être tout de même modélisé en associant ce bloc à l'étage d'entrée à transistors ("Input_q"). Soit G_d le gain de la source contrôlée modélisant le gain différentiel, C la capacité correspondant au pôle dominant et v_{amax} la valeur limite du signal différentiel de "Input_q", le slew-rate est alors égal à $S_r = \frac{G_d \cdot v_{amax}}{C}$, relation qui détermine la valeur de v_{amax}: $v_{amax} = S_r / K$, avec $K = 2 \cdot \pi \cdot f_p \cdot A_{vd}$. Or rappelons qu'une contrainte a été définie sur cette valeur lors de la description de "Input_q": $v_{amax} \geq 2 \cdot V_T$. Dans le cas où le gain différentiel de d'op-amp est supérieur à $A_{vdmax} = S_r / (4 \cdot \pi \cdot f_p \cdot V_T)$, il est nécessaire de décomposer la description du gain en deux étages: le premier étant l'étage "Gain" et le second "Gain_p". Dans notre cas, la valeur des deux gains est définie comme la racine carrée de Avd, si la condition précédente est vérifiée; la limitation de tension du premier étage est alors fixée par la relation $v_{amax1} = S_r / \left(2 \cdot \pi \cdot f_p \cdot \sqrt{A_{vd}} \right)$. Cette opération est réalisée automatiquement par le programme de génération des macro-modèles SPICE.

2.3 Bloc de sortie

Le bloc "Output" peut modéliser la résistance et la capacité de sortie ainsi que la limitation positive et négative du courant de sortie correspondant à une protection contre les courts-circuits. Cette caractéristique est réalisée à l'aide de diodes idéales de la même manière que pour le bloc "Gain_sr" et suivant le principe de *Connelly et al*. (cf Figure 33). De plus la structure symétrique de *Alexander et al.*, entre les sources d'alimentation s'est révélée indispensable. Le comportement obtenu est ainsi beaucoup plus proche de celui du circuit étudié. Le gain des sources contrôlées est égal à $G = 1/R$ avec $R = 2 \cdot R_{out}$. Si fp est le pôle modélisé alors $C = 1/(2 \cdot \pi \cdot f_p \cdot R)$. Signalons que pour la modélisation du circuit MC33272, le pôle de sortie n'a pas été décrit dans un premier temps mais sa prise en compte

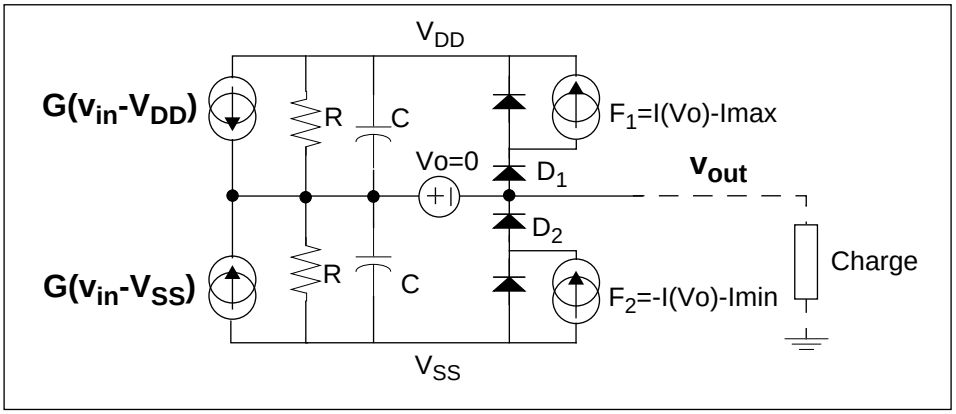


Figure 33 Étage SPICE de sortie de la bibliothèque

est envisagée dans l'avenir. Le code généré est le suivant:

```
.subckt Output in vdd vss out
go1 v0 vdd vdd in 5.882e-03
ro1 v0 vdd 1.700e+02
go2 vss 12 in vss 5.882e-03
ro2 v0 vss 1.700e+02
v0 v0 out dc 0
f1 1 vdd poly(1) v0 -3.700e-02 1.0
f2 vss 2 poly(1) v0 -3.700e-02 -1.0
d1 out 1 switch
d10 1 vdd switch
d2 2 out switch
d20 vss 2 switch
.model switch d n=0.001
.ends Output
```

3 Modèles comportementaux HDL-A

La modélisation comportementale de ces blocs de bas-niveau consiste en fait en l'écriture plus ou moins approchée des équations électriques des macro-modèles SPICE précédemment étudiés. Ainsi, dans le cas de l'étage d'entrée, il ne s'agit pas de réécrire le modèle du transistor mais plutôt de modéliser les principaux parasites. De même, concernant les limitations de tension ou de courant, les diodes seront remplacées par des fonctions linéaires par morceaux plus simples. Enfin, la fonction de transfert pourra être représentée par une seule équation différentielle. Les avantages apportés par cette approche sont principalement de réduire le nombre de composants afin de faciliter la convergence du simulateur et d'améliorer les temps de simulation pour une précision similaire. Ce paragraphe présente les techniques employées en HDL-A pour la modélisation des étages d'entrée, de gain et de sortie.

3.1 Équations d'entrée

L'architecture HDL-A qui a été développée est déduite du schéma fonctionnel de *Connelly et al*. (cf Figure 45c). Le modèle HDL-A est constitué d'une entité pour la déclaration des paramètres et des bornes de connexion, et d'une architecture pour la description des équations de Kirchhoff. Le code est le suivant:

```
ENTITY input IS
    GENERIC       (rcm,rdm,ios,ib,cin,vio : REAL) ;
    PIN           (inp,inm,dm,cm,gnd : ELECTRICAL) ;
END ENTITY ;

ARCHITECTURE ideal OF input IS
    CONSTANT ip,im : REAL;
    STATE vinp,vinm,va,icin,iinp,iinm : ANALOG ;
BEGIN
RELATION
        PROCEDURAL FOR INIT =>
             -- valeurs par défaut des paramètres:
             rcm := 1.000e+09;              -- résistance de mode commun
             rdm := 1.000e+06;              -- résistance differentielle
             ios := 0.0;                    -- courant de décalage
             ib := 1.0e-9;                  -- courant de polarisation
             cin := 1.0e-12;                -- capacité d'entrée
             vio := 0.0;                    -- tension d'offset
             -- deux constantes:
             ip := ib + ios/2.0;            -- source de courant sur inp
             im := ib - ios/2.0;            -- source de courant sur inm

        PROCEDURAL FOR DC, TRANSIENT, AC =>
             -- calcul des tensions de sortie:
             vinp := inp.V - vio;           -- entrée positive avec décalage
             vinm := inm.V;                 -- entrée negative
             va := vinp-vinm;               -- tension différentielle
             dm.V %= va;
             cm.V %= 0.5*(vinp+vinm);       -- tension de mode commun
             -- calcul des courants appliqués sur les bornes d'entrée:
             icin := cin*DDT(va);
             iinp := ip+(vinp/rcm)+(va/rdm)+icin; -- entrée positive
             iinm := im+(vinm/rcm)-(va/rdm)-icin; -- entrée negative
             inp.I %= iinp;
             inm.I %= iinm;
             gnd.I %= -iinp-iinm;
END RELATION ;
END ARCHITECTURE ;
```

Notons que la dynamique du signal différentiel, va, n'est pas limitée dans cet étage mais dans l'étage suivant de gain.

3.2 Étage principal de gain

Cet étage modélise le gain de mode différentiel et commun, les facteurs de réjection d'alimentation, le pôle dominant, la limitation de slew-rate et la limitation de tension. Il comporte deux entrées, une pour le signal différentiel (`d`) et une pour celui de mode commun (`cm`), deux bornes pour les alimentations (`vdd`, `vss`) et une sortie (`outp`):

```
ENTITY gain IS
    GENERIC       (avd,p1,cmrr,ppsrr,npsrr,srp,srn,dvp,dvm: REAL) ;
    PIN           (d,cm,vdd,vss,outp: ELECTRICAL) ;
END ENTITY ;

ARCHITECTURE ideal OF gain IS
    CONSTANT vp,vm,t1,k: REAL;
    STATE vg,vout,vmid,vmax,vmin,flim: ANALOG ;
BEGIN
RELATION
        PROCEDURAL FOR INIT =>
             -- valeurs par défaut des paramètres:
             avd := 1.0e+5;            -- gain différentiel
             p1 := 1.0e+2;             -- fréquence de coupure
             cmrr := 1.0e+6;           -- facteur de réjection de mode commun
             ppsrr := 1.0e+6           -- réjection d'alimentation positive
             npsrr := 1.0e+6           -- réjection d'alimentation négative
             srp := 1.0e6;             -- slew-rate positif
             srn := 1.0e6;             -- slew-rate négatif
             dvp:=0.001;               -- coeff. de saturation maxi.
             dvm:=0.001;               -- coeff. de saturation mini.
             -- constantes:
             k := avd;                 -- facteur de limitation
             t1 := 1.0/(twopi*p1);     -- constante de temps
             vp := srp*t1;             -- limite maxi. pour srp
             vm := -abs(srn)*t1;       -- limite mini. pour srn

        PROCEDURAL FOR DC, TRANSIENT, AC =>
             -- tension amplifiée:
             vg := avd*(d.V + cm.V/cmrr - vdd.V/ppsrr + vss.V/npsrr);
             -- limitation de la dynamique pour le slew-rate:
             IF vg>=vp THEN vg:=vp;
             ELSIF vg<=vm THEN vg:=vm;
             END IF;
             -- calcul des tensions maxi. et mini.:
             vmid:=(vdd.V+vss.V)/2.0;            -- point milieu
             vmax:=vdd.V*(1.0-dvp)-vmid;         -- tension maxi. >0
             vmin:=-(vmid-vss.V*(1.0-dvm));      -- tension mini. <0
             -- fonction de limitation de la tension de sortie:
             IF vout>=vmax THEN        flim:=k*(vout-vmax);
             ELSIF vout<=vmin THEN     flim:=k*(vout-vmin);
             ELSE                      flim:=0.0;
             END IF;
```

```
            -- application de la tension de sortie:
            outp.V %= vout;

        EQUATION(vout) FOR DC, AC, TRANSIENT =>
            -- équation différentielle d'un filtre passe-bas & limitation:
            t1*DDT(vout) + vout - vg + flim == 0.0;
END RELATION ;
END ARCHITECTURE ;
```

L'instruction **`vg := avd*(d.V+cm.V/cmrr-vdd.V/ppsrr+vss.V/npsrr);`** permet de calculer le signal `vg` dont la dynamique est ensuite limitée par une instruction conditionnelle pour modéliser les slew-rates positifs et négatifs, à la manière du macro-modèle de *Chua et al.* [ChL75] (cf Figure 34), qui utilise une transconductance non-linéaire. Dans le modèle HDL-A, la tension maximale est définie par $v_p = t_1 \cdot sr_p$ et la tension minimale par $v_m = -t_1 \cdot sr_n$, où t_1 est la constante de temps correspondant au pôle de l'étage, sr_p et sr_n les *slew-rates* des fronts positifs et négatifs.

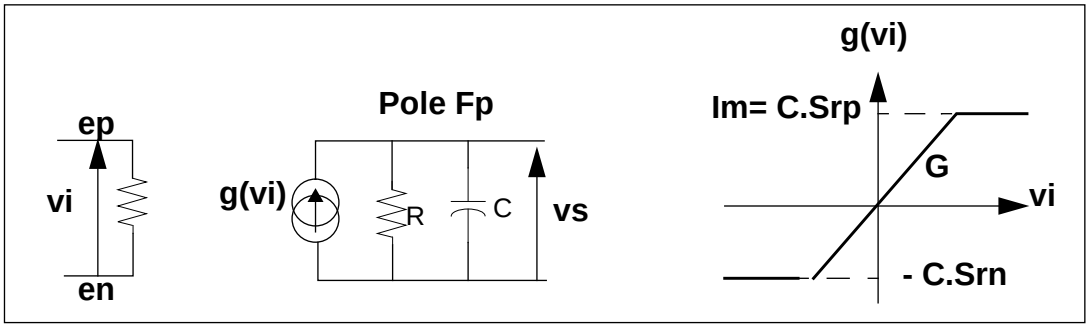


Figure 34 Modélisation du slew-rate par une transconductance non-linéaire [ChL75]

Le filtre passe-bas et la limitation de tension sont modélisés par l'équation différentielle non-linéaire, placée dans le bloc équation d'inconnue v_{out}:

$t_1 \cdot \frac{dv_{out}}{dt} + v_{out} - v_g + f_{lim}(v_{out}) = 0$, où $f_{lim}()$ est une fonction linéaire par morceaux permettant de réaliser la limitation de l'inconnue v_{out} (fonction **pwl** de la Figure 35). En effet, le terme f_{lim} devient prépondérant lorsque v_{out} dépasse les valeurs limites *vmax* et *vmin* ce qui réalise en quelque sorte une contrainte sur v_{out}.

Signalons que cette méthode s'inspire des équations d'un schéma de limitation à l'aide de diodes, qui ont été aussi modélisées sous la forme $f_{lim}(v) = is \cdot \left(e^{k \cdot (v - v_{max})} - e^{k \cdot (v_{min} - v)} \right)$ où *is* correspond au courant de saturation inverse et *k* à l'inverse de la tension de seuil (courbe **exp** de la Figure 35). Cependant, un certain nombre de contrôles doivent être alors réalisés pour éviter toute erreur de débordement mathématique des fonctions exponentielles et une fonction externe C a été développé pour ne pas alourdir le code HDL-A. On se reportera à la

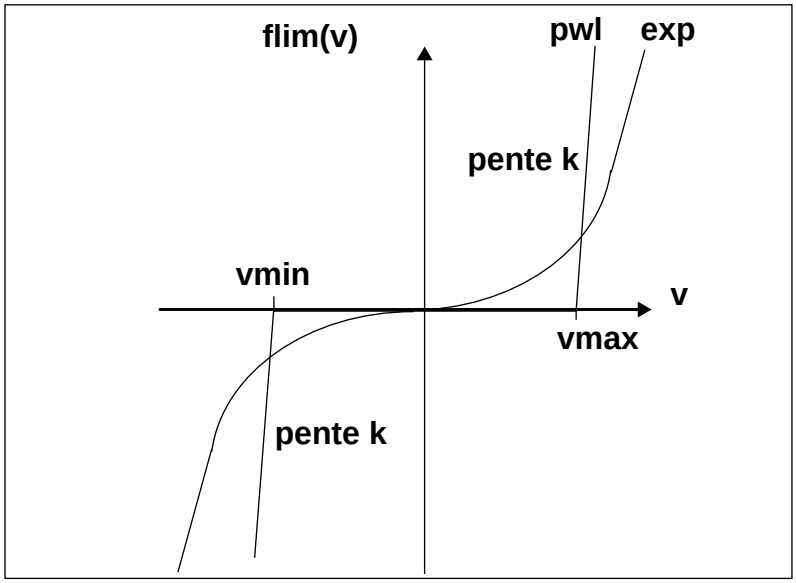


Figure 35 Fonctions de limitation de *v* entre *vmax*, *vmin*

modélisation de l'étage de sortie au paragraphe suivant pour un exemple d'une telle fonction. Notons enfin que l'approche qui consisterait à scinder l'équation différentielle avec contrainte, décrite précédemment, en une équation différentielle linéaire $t_1 \cdot \frac{dv_1}{dt} + v_1 - v_g = 0$, d'inconnue une variable analogique v_1, et en une instruction `if-then-else` de limitation de la tension v_1 pour le calcul de la tension de sortie v_{out}, conduit à des résultats de simulation transitoire qui sont erronés car la tension intermédiaire v_1 peut atteindre des tensions physiquement aberrantes.

3.3 Équations de sortie

Le modèle HDL-A de l'étage de sortie consiste en la description d'une source de tension de résistance `rout`, d'une limitation du courant de court-circuit et enfin d'une limitation de la tension de sortie en fonctionnement proche du court-circuit. Le code HDL-A du modèle de l'étage de sortie est le suivant:

```
ENTITY output IS
    GENERIC       (rout,imax,imin,dvp,dvm,iis,vth: REAL) ;
    PIN           (inp,vdd,vss,outp : ELECTRICAL) ;
END ENTITY ;
ARCHITECTURE ideal OF output IS
    CONSTANT isource, isink: REAL;
    STATE vmid,vmax,vmin,vin: ANALOG;
    STATE iout,ip,im,isatp,isatm: ANALOG ;
    FUNCTION vlim(CONSTANT a,b: REAL; c: ANALOG) RETURN ANALOG;
BEGIN
```

```
    RELATION
        PROCEDURAL FOR INIT =>
            rout := 1.0e+2;           -- résistance de sortie
            imax:=0.1;                -- courant maxi.
            imin:=0.1;                -- courant mini.
            dvp := 0.001;             -- coeff. de saturation max.
            dvm := 0.001;             -- coeff. de saturation min.
            iis := 1.0e-12;           -- courant de saturation inverse
            vth := 26.0e-03;          -- tension de seuil
            isource := -abs(imax);
            isink := abs(imin);

        PROCEDURAL FOR DC, AC, TRANSIENT =>
            vmid:= (vdd.V+vss.V)/2.0;           -- point milieu
            vin:= inp.V+vmid;                   -- tension d'entrée
            vmax:=vdd.V*(1.0-dvp)-vmid;         -- tension maximale
            vmin:=-(vmid-vss.V*(1.0-dvm));      -- tension minimale
            -- Limitation du courant de sortie:
            IF iout<=isource THEN ip := -iout+isource;
            ELSIF iout>=isink THEN im := -iout+isink;
            ELSE ip := 0.0; im := 0.0;
            END IF;
            -- Limitation de la tension outp.V entre vmax et vmin:
            idp := vlim(iis,vth,outp.V-vmax); -- diode Dp
            idm := -vlim(iis,vth,vmin-outp.V); -- diode Dn
            -- Application des courants sur outp, vdd, vss:
            outp.I %= iout+ip+im+idp+idm;       -- Loi des noeuds en outp
            vdd.I %= -iout/2.0-ip-idp;          -- Loi des noeuds en vdd
            vss.I %= -iout/2.0-im-idm;          -- Loi des noeuds en vss

        EQUATION(iout) FOR DC, AC, TRANSIENT =>
            -- Calcul du courant iout:
            rout*iout == outp.V - vin ;
        END RELATION ;
END ARCHITECTURE ideal;
```

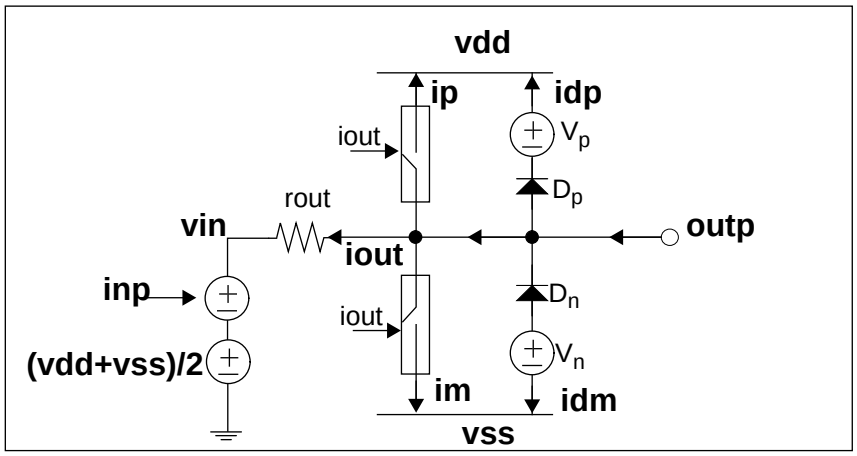


Figure 36 Schéma équivalent de l'étage de sortie HDL-A

Le courant traversant la source de tension est défini comme l'inconnue `iout` de l'équation implicite: `rout*iout == outp.V - vin ;` où `outp.V` est la tension de sortie, `vin` la tension provenant de l'étage précédent (défini par rapport au point milieu des alimentations `vmid`) et `rout` la résistance de sortie. Le schéma de la Figure 36 montre le sens de `iout`. L'utilisation d'une telle équation est indispensable pour faciliter la convergence du simulateur.

Le courant appliqué sur la borne de sortie, `outp`, est la somme de ce courant `iout`, des courants `ip` et `im`, et des courants `idp` et `idm`:

- `ip` et `im` sont définis comme l'excès de courant de `iout` par rapport aux valeurs limites respectives `isource` et `isink`, et sont appliqués entre `outp` et les bornes d'alimentation selon le schéma de la Figure 36.
- `idp` et `idm` correspondent aux courants des diodes qui seraient placées entre la borne de sortie et les bornes d'alimentation, comme illustré en Figure 36, afin d'effectuer une limitation de la tension de sortie en cas de court-circuit. Ils sont calculés à l'aide de la fonction *C* `vlim` modélisant la caractéristique exponentielle `I(V)` d'une diode. La fonction `vlim` a été écrite en utilisant les fonctions de l'interface C de HDL-A pour le passage des arguments. Elle est déclarée dans le modèle HDL-A par l'instruction:

```
FUNCTION vlim(CONSTANT a,b: REAL; c: ANALOG) RETURN ANALOG;
```

où `a` et `b` sont les coefficients correspondant respectivement au courant de saturation inverse et au seuil d'une diode, et `c` est la variable analogique représentant la tension aux bornes de la diode. Le code de la fonction *C-HDL-A* est le suivant:

```
#include <math.h>
#include "fci.h"

void VLIM (       returnType, returnValue, Type1, Value1,
                  Type2, Value2, TypeV, ValueV )
fci_type * returnType; fci_value * returnValue;
fci_type * Type1; fci_value * Value1;
fci_type * Type2; fci_value * Value2;
fci_type * TypeV; fci_value * ValueV;
{
        double x = 200.0;              /* limite sup. de x pour exp(x) */
        double vm = -1.0;              /* seuil pour l'asymptote */
        double gmin = 1.0e-12;         /* conductance */
        double is, vt, v, vp, i;
        double a, b;                   /* coefficients de la tangente */

        is = fci_get_real (Value1);    /* courant de saturation inverse */
        vt = fci_get_real (Value2);    /* tension de seuil */
        v = fci_get_real (ValueV);     /* tension */
```

```
        vp = x*vt;                      /* seuil pour la tangente */
        a = is*exp(x)/vt + gmin;        /* tangente y = a*x+b en x=200 */
        b = is*(exp(x)-1.0)-vp*a;
        if (v>=vp) i = a*v+b;                           /* tangente */
        else if (v>=vm) i = is*(exp(v/vt)-1.0);         /* exp. */
        else i = -is;                                   /* asymptote */
        fci_set_analog (ValueV, i+v*gmin, 0.0);
}
```

Pour simplification, les contrôles de type des divers arguments de la fonction ne sont pas décrits. Les valeurs des paramètres réels sont obtenues par la fonction `fci_get_real`() et la définition de la valeur de retour, analogique, est réalisée par la fonction `fci_set_analog()`. La Figure 37 illustre la fonction modélisée, qui se compose de trois parties:

- une asymptote `i=-is` pour des tensions `v` inférieures à `-1.0`,
- une tangente d'équation `i=a*v+b` pour des tensions `v` supérieures à `200.0*vt`, afin d'éviter les erreurs de débordement mathématique de la fonction exponentielle `exp`,
- la caractéristique `i(v)` d'une diode `i=is*(exp(v/vt)-1.0)` entre les deux limites précédentes.

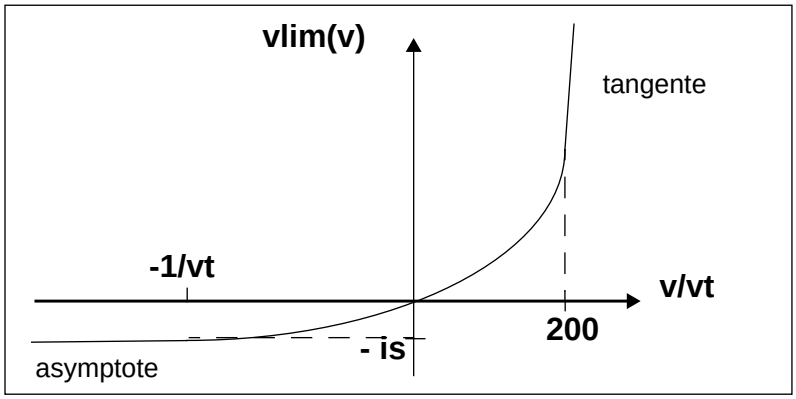


Figure 37 Modèle d'une diode de seuil *vt* et de courant de saturation inverse *is*

4 Application et validation: op-amp *MC33272*

4.1 Structure du macro-modèle

Les deux approches de modélisation, macro-modélisation SPICE et modélisation comportementale HDL-A, ont été appliquées à un nouvel op-amp à vitesse et précision élevée de SGS-Thomson, de référence *MC33272*. Ses performances, ainsi que celles des macro-modèles, ont été mesurées à l'aide de l'outil de caractérisation qui sera décrit au Chapitre 6.

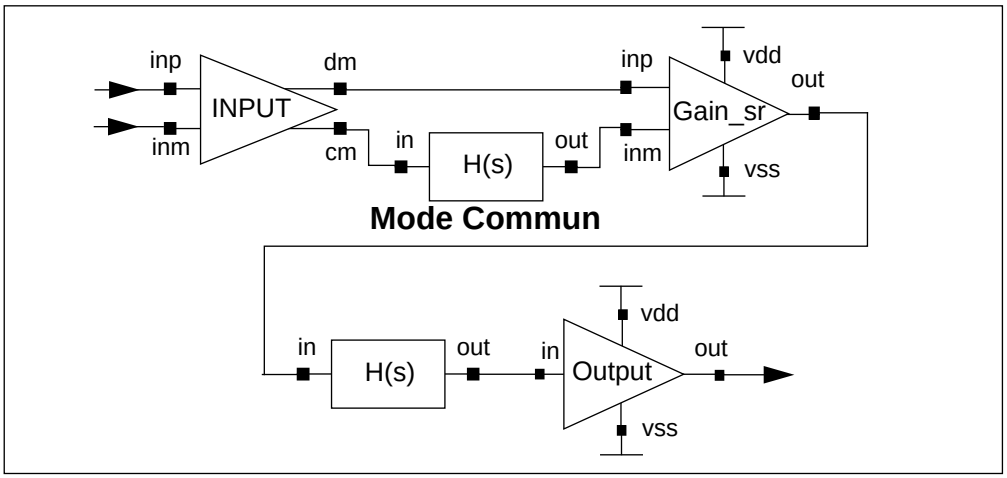


Figure 38 Schéma du macro-modèle HDL-A du circuit *MC33272*

La Figure 38 définit ainsi le schéma d'un des macro-modèles étudiés et constitué des blocs HDL-A *Input*, *Gain_sr*, *H(s)* et *Output*. Notons que deux blocs de Laplace *H(s)* sont utilisés, l'un décrivant les racines spécifiques de la fonction de transfert en mode commun (deux zéros et un pôle) et l'autre les racines secondaires de la fonction de transfert en mode différentiel (cinq pôles et un zéro).

Deux structures de macro-modèle SPICE ont été d'autre part considérées:

- la première est basée sur les blocs symétriques proposés par *Alexander et al*. *Input_q*, *Gain_p*, *Pole_sym*, *Zero_sym* et *Output*. Plusieurs étages de pôles et zéros sont en fait utilisés pour décrire les différentes racines correspondant aux blocs précédents *H(s)*.
- la seconde fait appel aux blocs de *Connelly et al*, *Input*, *Gain_sr*, *Pole*, *Zero et PoleZero* et à l'étage symétrique de sortie *Output*. Les mêmes racines ont été prises en compte.

4.2 Résultats de simulation - Caractérisation

Le Tableau 12 permet de comparer les caractéristiques du circuit, des deux types de macro-modèle SPICE et du macro-modèle HDL-A. Précisons que les schémas de mesure sont définis en Annexe 6. On peut remarquer que les modèles sont assez précis, tant pour les caractéristiques DC que temporelles et fréquentielles.

Les résultats de simulation ***transitoire*** du circuit, du macro-modèle SPICE de type *Alexander et al.* et du modèle HDL-A sont représentés en Figure 39, la réponse du macro-modèle SPICE de type *Connelly et al.* étant pratiquement identique à celle du modèle HDL-A. Notons que pour le front ascendant, le macro-modèle SPICE qui utilise un étage différentiel

Tableau 12 *Datasheet* du circuit et des modèles SPICE et HDL-A

Caractéristique	*Circuit MC33272*	*Macro SPICE Alexander*	*Macro SPICE Connelly et al*	*Modèle HDL-A*
Tension Maximum	14.4 V	14.4 V	14.4 V	14.4 V
Tension Minimum	-14.8 V	-14.8 V	-14.8 V	-14.8 V
Tension de décalage	0.057 µV	8 µV	0.056 µV	0.057 µV
Courant bornes d'entrée	0.091 µA	0.091 µA	0.091 µA	0.091 µA
Slew-rate positif	19 V/µs	18.6 V/µs	15.6 V/µs	16 V/µs
Slew-rate négatif	-12 V/µs	-11 V/µs	-10.2 V/µs	-10.7 V/µs
Gain différentiel	97.3 dB	97 dB	97 dB	97 dB
CMRR	119.8 dB	119.8 dB	119.8 dB	119.8 dB
Gain-Bande à 0 dB	4.9 MHz	4.6 MHz	4.6 MHz	4.7 MHz
Marge de gain	-8.2 dB	- 14 dB	-10.6 dB	-9.5 dB
Marge de phase	37.2 degrés	37.5 degrés	37.5 degrés	35.2 degrés
Résistance de sortie	85 Ohm	85 Ohm	85 Ohm	85 Ohm
Courant de sortie Max.	37 mA	37 mA	37 mA	37 mA

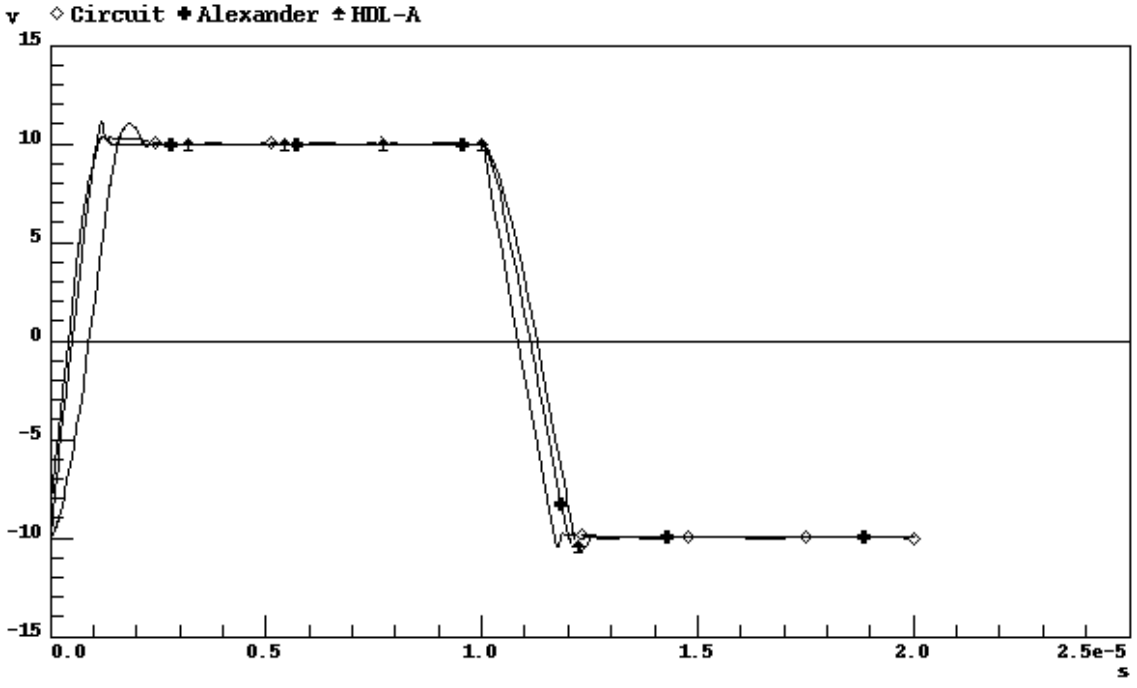


Figure 39 Réponses transitoires du circuit, des modèles SPICE et HDL-A

d'entrée (modèle de *Alexander et al.*) permet d'obtenir une plus grande précision. Notons d'autre part que les résultats sont corrects même si l'amplitude de la tension d'entrée est proche des tensions d'alimentation.

Concernant la réponse ***AC*** de mode différentiel, les modèles ont été conçus pour être précis jusqu'à environ 200 MHz, c'est à dire au delà de la bande passante du circuit (cf Figure 40). Notons que la marge de phase, paramètre très important pour l'étude de la stabilité d'un amplificateur en boucle fermée, est correctement modélisée. Quant à la fonction de transfert de mode commun, la modélisation n'est précise que jusqu'à environ 100 kHz (cf Figure 41). Une précision plus élevée pourrait être simplement obtenue en augmentant le nombre de racines de la fonction de transfert. Cependant, un compromis entre précision et temps de simulation doit être effectué.

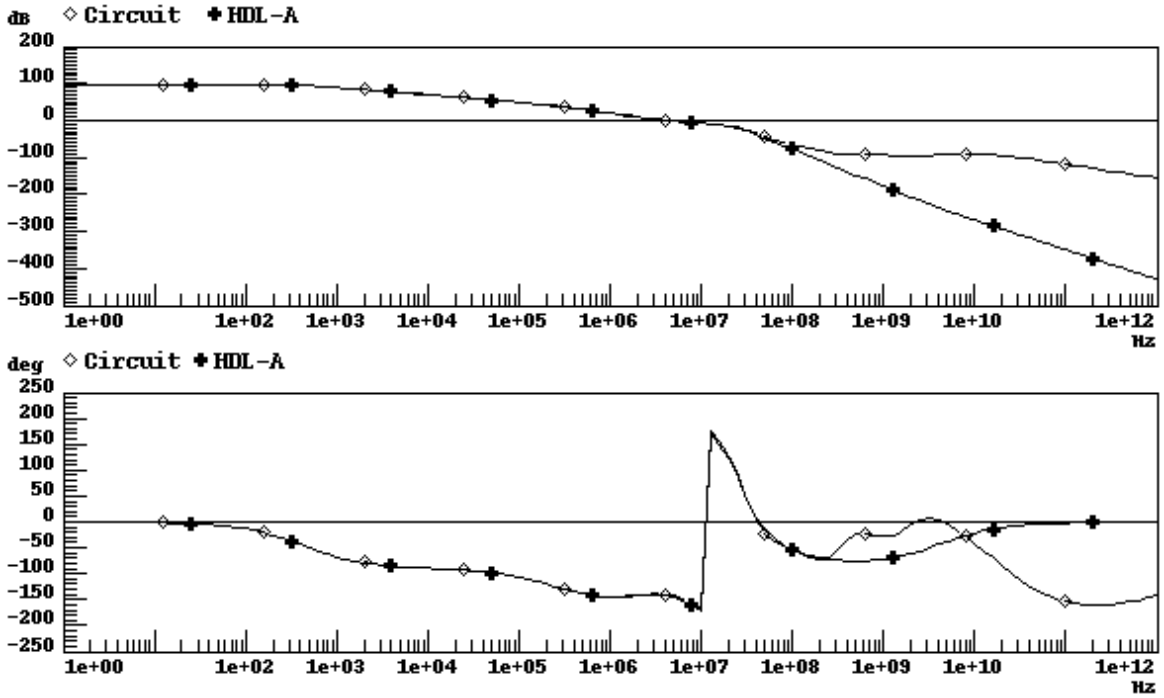


Figure 40 Gain et Phase de la sortie différentielle

Enfin, la Figure 42 met en évidence la modélisation de la résistance de sortie et des limitations de courant, l'op-amp considéré étant étudié en suiveur dans des conditions proches de la saturation (le schéma de mesure est exposé en Annexe 6). Une tension de 13 V est ainsi appliquée en entrée pour la mesure du courant maximum, dit *Isource*, et une tension de -13 V pour la mesure du courant minimal, dit *Isink*. La résistance de sortie du circuit *MC33272* a été relevée dans la zone où la tension de sortie varie linéairement en fonction du courant de charge et donc avant le décrochement dû aux limitations de courant à 37mA.

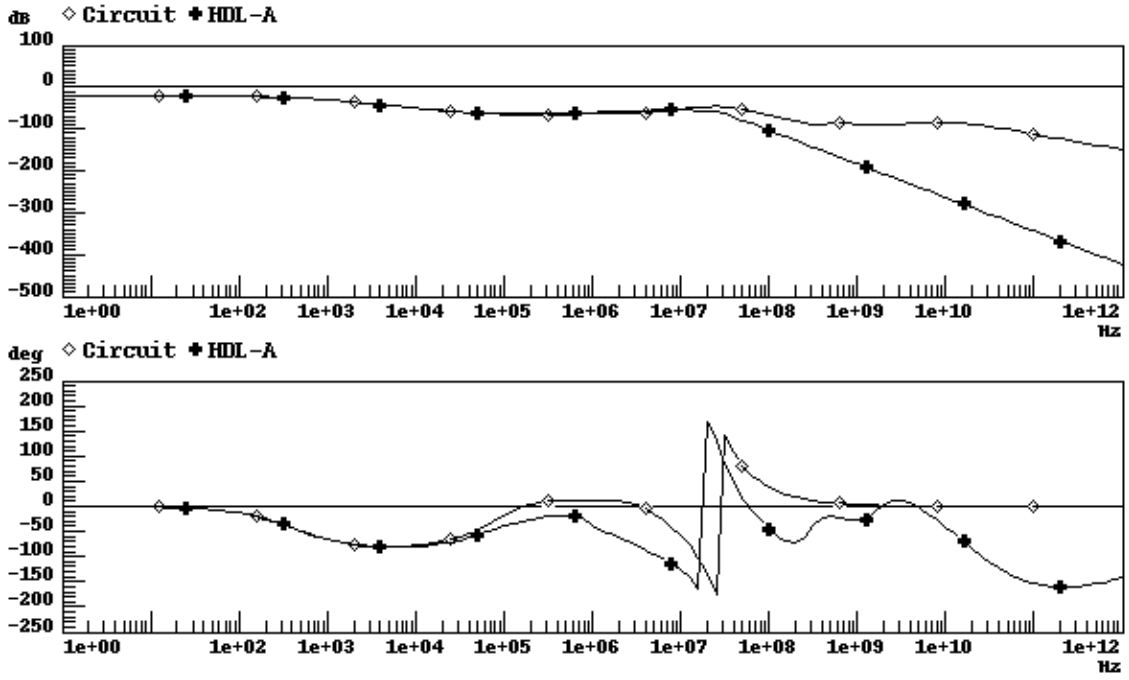


Figure 41 Gain et Phase de la sortie de mode commun

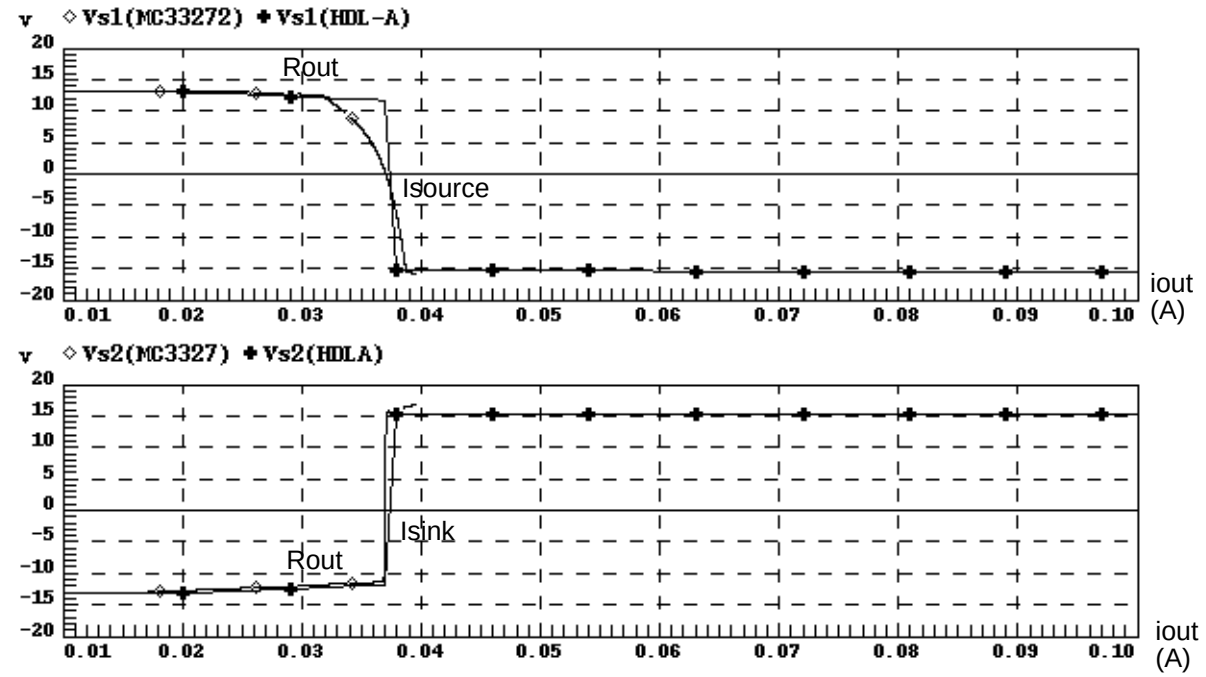


Figure 42 Mesure des courants de court-circuit *Isource* et *Isink*

5 Conclusion

Les blocs SPICE ou HDL-A de la bibliothèque d'étages élémentaires ont donc permis de décrire assez finement les diverses caractéristiques d'un nouvel amplificateur opérationnel large bande et de précision élevée de SGS-Thomson. Rappelons que les paramètres des

modèles ont été obtenus à l'aide de l'outil de caractérisation présenté au Chapitre 6 et qui effectue une série de simulations du circuit. Les macro-modèles qui ont été développés, SPICE et HDL-A, devront bien sûr être raffinés à la suite de la réalisation physique du circuit à partir dans ce cas, des mesures des performances réelles.

Enfin, les différents étages de la bibliothèque pourraient être améliorés en prenant en compte diverses non-linéarités, par exemple celles du gain ou de la résistance de sortie, le bruit des composants, ainsi que les variations statistiques des paramètres technologiques par une caractérisation des distributions statistiques des performances du circuit.

Chapitre 6: Caractérisation de Circuits Analogiques

1 Les objectifs

Un certain nombre d'outils de caractérisation analogique ont été cités au Chapitre 3: *Anacar* [HGT91], *Simboy* [NHM94][HHN94], ainsi qu'un outil interne à SGS-Thomson [ACT92]. Ils permettent de générer des tables de performance ou data-sheets, à partir desquelles on peut calculer les paramètres de macro-modèles. Cependant, le lien entre caractérisation et macro-modélisation n'est pas immédiat. Notre premier objectif a donc été d'élaborer un système de caractérisation qui reprenne les caractéristiques principales des outils étudiés et qui puisse être réellement dirigé par les besoins de la macro-modélisation, les paramètres des macro-modèles devant être calculés automatiquement.

Enfin, un tel environnement doit être facilement paramétrable, adaptable aux besoins spécifiques des divers utilisateurs et le plus interactif possible. C'est pourquoi nous avons choisi de l'intégrer dans l'environnement de conception *"Design Framework II"* [DFR94], qui offre de nombreuses facilités pour le développement d'interfaces humaines.

2 Le principe de l'environnement

Il est présenté en Figure 43. Cet environnement possède deux entrées graphiques:

- un ***schéma bloc définissant un macro-modèle*** dont les paramètres sont à déterminer en fonction des performances du circuit qu'il doit représenter. Le schéma est établi à partir de la bibliothèque de modèles fonctionnels de bas-niveau qui a été présentée au chapitre précédent. De plus, les paramètres des différents blocs sont définis en fonction des noms des mesures disponibles dans la bibliothèque de caractérisation associée à l'environnement, à l'aide d'un nouveau *langage de spécification de mesures*.
- un ***schéma d'appel de différentes procédures de caractérisation*** pour la simple génération de *data-sheet*. Cette seconde entrée permet de vérifier le comportement d'un macro-modèle.

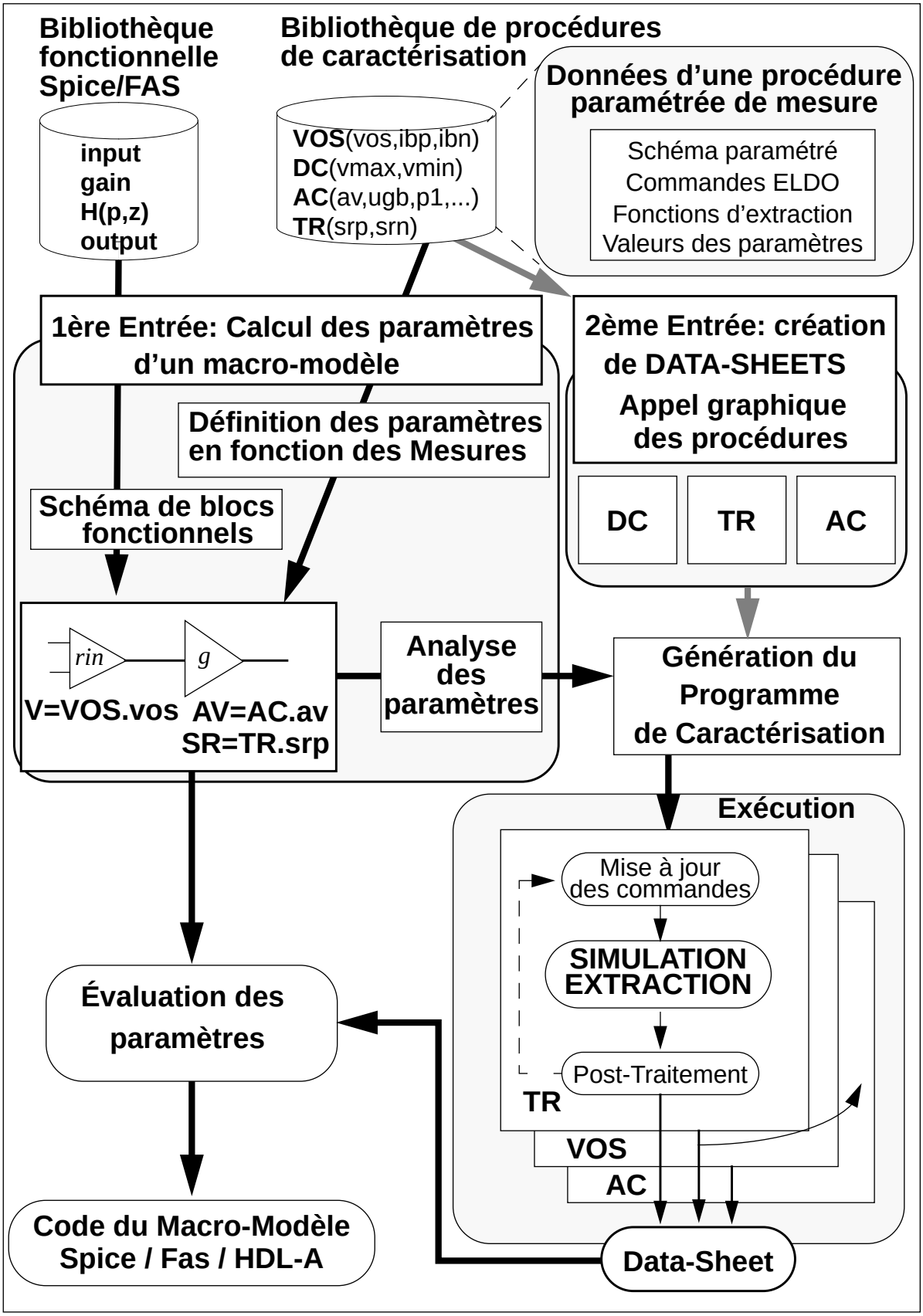


Figure 43 Diagramme synoptique de l'outil

Rappelons qu'un système de caractérisation consiste en un ensemble de ***procédures de mesure*** qui font entre autres appel à un simulateur pour générer les courbes, à partir desquelles sont extraites les propriétés du circuit.

Ces procédures doivent être au préalable stockées dans une bibliothèque de caractérisation. Actuellement, la bibliothèque associée au prototype concerne seulement les op-amps, mais l'environnement offre à l'utilisateur la possibilité de définir facilement ses propres procédures de mesure. Les données à spécifier comprennent le schéma de mesure ou *test-bench*, les commandes de simulation, les fonctions d'analyse des courbes, appelées fonctions d'*extraction*, les programmes éventuels de post-traitement et le mode de lancement de la simulation, qui peut être itératif pour résoudre des problèmes de mesure. De plus, ces procédures sont entièrement *paramétrables* dans un but de généralité et il est en outre possible de définir un paramètre en fonction du résultat d'une autre procédure.

Les procédures sont exécutées selon un plan appelé ***programme de caractérisation***, qui prend en compte la notion précédente de hiérarchie. Concernant la première entrée de l'outil, le plan est automatiquement généré par analyse des expressions des paramètres de blocs. Après exécution de la caractérisation proprement dite, ces expressions sont utilisées pour calculer numériquement les paramètres et permettre la génération du code qui correspond au schéma bloc (en langages SPICE, FAS ou HDL-A). Concernant la seconde entrée, le plan est établi en fonction des procédures définies sur le schéma et implicitement appelées dans l'expression de leurs paramètres. Mais avant de détailler ces deux types d'utilisation de l'outil de caractérisation, précisons les données qui doivent être associées à une procédure.

3 Définition des données d'une procédure de mesure

3.1 Le schéma de mesure et les commandes de simulation

Chaque procédure a son propre schéma de mesure. La caractérisation du gain d'un amplificateur est par exemple réalisée en boucle ouverte, alors que celle du *slew-rate* est effectuée avec un amplificateur en boucle fermée (cf Figure 44). Le schéma est saisi par l'éditeur graphique de l'environnement de conception en utilisant les éléments d'une bibliothèque fonctionnelle qui comporte les symboles de toutes les primitives du simulateur. Les commandes de simulation (analyse DC, transitoire, fréquentielle, analyse des pôles et

zéros, analyse du bruit, etc...), la liste des courbes à analyser, les valeurs des alimentations, etc..., sont quant à elles saisies par une interface développée à SGS-Thomson dans le cadre de l'*Environnement Analogique de Conception* [MLN93].

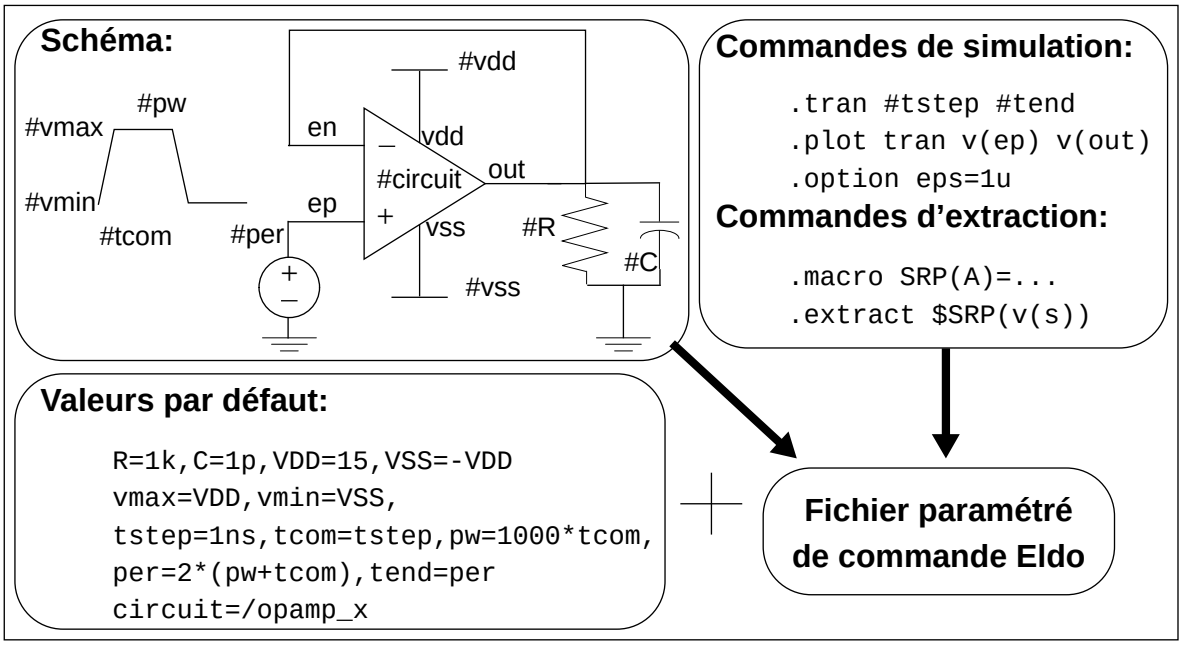


Figure 44 Schéma de mesure pour l'analyse transitoire d'un op-amp

Dans un but de généralité, toutes les données sont ***paramétrables***: pour identifier les conditions ou paramètres de la mesure, une syntaxe particulière doit être utilisée. On peut simplement faire précéder le nom du paramètre d'un caractère spécial (ici #). Par exemple, la valeur de la résistance de charge de l'op-amp de la Figure 44 est définie par `#R` lors du placement du composant sur le schéma. De même, la source impulsionnelle d'entrée est définie par: `VEP EP 0 PULSE(#vmin #vmax 0 #tcom #tcom #pw #per)` [Eld94]...

Enfin, un schéma de mesure est applicable à différents circuits d'une même classe. Dans le cas de la Figure 44, il s'agit d'op-amps à sortie asymétrique. Concernant le prototype, la variable `#circuit` est ainsi définie pour préciser le chemin d'accès au fichier de description du circuit étudié. Ce dernier est inclus dans le fichier de commande du simulateur par l'intermédiaire de la commande "`.INCLUDE #circuit`" qui est placée dans un sous-circuit pour assurer la cohérence des points de connexion avec le symbole d'op-amp du schéma:

```
.SUBCKT OPA EP EN VDD VSS OUT
.INCLUDE #circuit
.ENDS OPA
```

où "`EP EN VDD VSS OUT`" définit la liste des points de connexion, et **`OPA`** est le nom du symbole. Notons que les paramètres des modèles de transistors utilisés par le circuit doivent être

décrits dans ce même fichier, à moins d'en inclure un autre.

3.2 Les commandes d'analyse des courbes

Ces commandes ont pour but de calculer les performances du circuit étudié à partir des résultats de simulation. Par exemple, à la suite d'une analyse fréquentielle d'un amplificateur en boucle ouverte, on peut déterminer la marge de gain et la marge de phase à partir des courbes de gain et de phase de la sortie. Certains simulateurs, comme Eldo [Eld94], possèdent un langage et une bibliothèque de fonctions de traitement des résultats, aussi nommées fonctions d'*extraction*. Elles permettent à l'utilisateur de créer des fonctions plus complexes, nommées ***macros***. Les fonctions Eldo sont actuellement utilisées par l'outil de caractérisation par souci de simplification, mais d'autres outils de manipulation de courbes existent, comme *Pluto*[Plu92] ou *SimPilot*[Sim94], dont l'utilisation est d'ailleurs envisagée pour l'avenir.

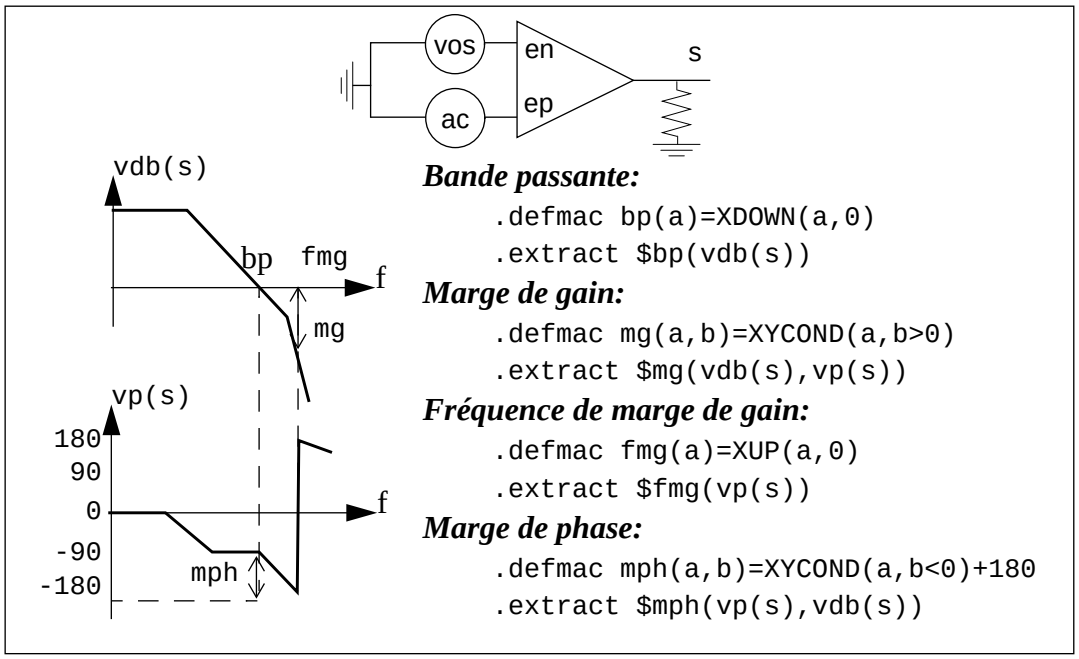


Figure 45 Mesure des performances fréquentielles d'un op-amp

Un exemple d'utilisation des fonctions Eldo pour la caractérisation de la réponse fréquentielle d'un op-amp est présenté en Figure 45. Les fonctions génériques mesurant la bande passante, la marge de gain, la fréquence correspondant à la marge de gain et enfin la marge de phase sont définies par la commande `.DEFMAC` [Eld94]. Ainsi, les macro-fonctions `BP`, `MG`, `FMG` et `MPH` sont construites grâce aux fonctions de base `XUP`, `XDOWN` et `XYCOND` qui permettent de déterminer l'abscisse de la courbe lorsque respectivement un seuil est franchi de

manière ascendante, descendante et lorsqu'une condition booléenne est réalisée. Les *macros* sont alors appelées par la commande `.EXTRACT` et ont pour paramètres les courbes de gain `VDB(s)` et de phase `VP(s)`, où `s` est le noeud de mesure. Ces commandes sont intégrées au fichier de commande du simulateur.

Notons qu'une interface a été développée pour faciliter l'écriture de telles macro-fonctions, en présentant la liste des fonctions de base disponibles ainsi que la définition associée (cf paragraphe 6). L'interface permet aussi d'assurer la cohérence du nom des courbes du schéma.

3.3 Valeurs par défaut des conditions de simulation

La liste des paramètres de la procédure en cours d'élaboration est automatiquement obtenue par analyse du fichier de commande du simulateur, qui a été généré par l'environnement de conception analogique (rappelons que les noms des paramètres sont précédés d'un caractère spécial). Une valeur par défaut doit être définie par l'utilisateur. Ce peut être une valeur numérique ou une expression rendant une valeur numérique ou encore une chaîne de caractères pour préciser le chemin d'accès au fichier du circuit étudié. Il est d'autre part possible de définir deux autres types de relations:

- ***entre paramètres d'une même procédure***: par exemple, la valeur de l'alimentation négative est l'opposée de l'alimentation positive,
- ***entre les paramètres d'une procédure et des résultats de caractérisation d'autres procédures***: par exemple, dans le cas de l'analyse fréquentielle d'un op-amp en boucle ouverte, le schéma comporte en entrée une source de tension qui compense la tension parasite de décalage (*offset*), et dont la valeur dépend d'une analyse DC du circuit avec un autre schéma de mesure. Notons que les deux procédures qui sont ainsi liées doivent présenter une certaine cohérence quant aux conditions de mesure: par exemple, il faudra caractériser le circuit à la même température dans les deux cas. Il est donc indispensable de pouvoir préciser un jeu spécifique de paramètres pour une procédure, nommé ***configuration***.

Pour décrire de telles relations, un ***langage de spécification de mesures*** a été défini. Sa grammaire BNF (acronyme de *Backus-Naur Form*) est basée sur une grammaire classique des expressions algébriques mais comporte deux entrées en plus: **`fichier`**, correspondant au nom

du fichier de description du circuit étudié, et **`paramètre`** pour définir une mesure:

```
expression ::= fichier | algébrique
fichier ::= "<chemin_accès>"
algébrique ::= [signe] terme { opérateur_add terme }
signe ::= + | -
opérateur_add ::= + | -
terme ::= facteur { opérateur_mult facteur }
opérateur_mult ::= * | /
facteur ::= nombre | paramètre | (algébrique)
nombre ::= <réel>échelle
échelle ::= F | P | N | U | M | K | MEG | G | T
              | f | p | n | u | m | k | meg | g | t
paramètre ::= analyse . caractéristique
analyse ::= <nom>[@configuration]
configuration ::= <nom>
caractéristique ::= <nom> [défaut]
défaut ::= '[' nombre ']'
```

où `<chemin_accès>`, `<nom>` et `<réel>` désignent respectivement une chaine de caractère spécifiant l'accès à un fichier, un identificateur et un nombre réel, le symbole | une alternative, les crochets [] une règle optionnelle et les parenthèses {} une règle optionnelle multiple. Soit par exemple les relations suivantes pour les paramètres d'une procédure nommée AC effectuant une analyse fréquentielle:

```
vdd = 3.3
vss = -AC.vdd
voffset = DC@C1.vos[1u]
circuit = "/usr/opamp/description1.mod"
```

`vdd` est une constante numérique, à partir de laquelle `vss` est définie. Le paramètre `voffset` est égal à la caractéristique `vos` de la procédure `DC` (externe) dans la configuration `C1`. Cependant, lors de la phase d'élaboration de la procédure `AC`, la valeur spécifiée entre crochets, `1u`, est utilisée. Enfin, `circuit` indique le chemin d'accès à la *netlist* du circuit étudié.

3.4 Différents types de procédure

La simulation est lancée par une procédure qui effectue un certain nombre d'actions successives (cf Figure 46):

- l'évaluation des paramètres de la procédure en fonction des valeurs spécifiées par l'utilisateur avant le lancement de la caractérisation ainsi que des résultats d'autres

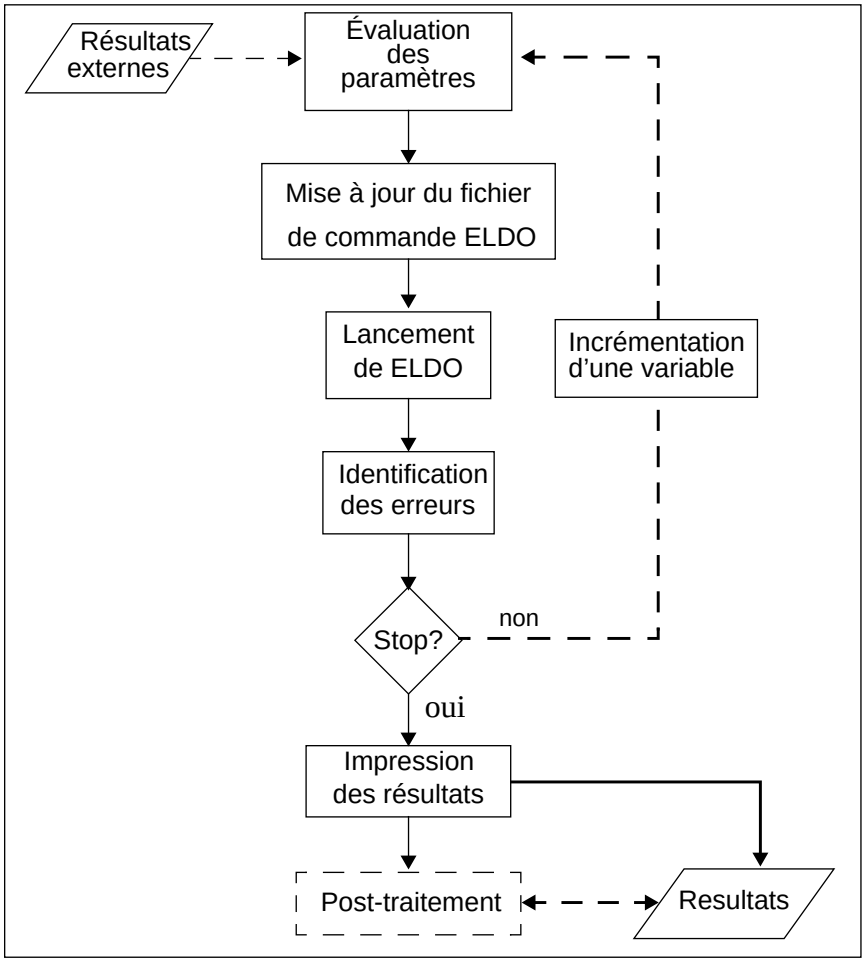


Figure 46 Les étapes d'une procédure de caractérisation

procédures,

- la mise à jour du fichier paramétré de commande du simulateur: ce pré-traitement effectue le remplacement des paramètres par leurs valeurs effectives précédemment calculées,
- le lancement du simulateur Eldo qui détermine, grâce aux fonctions d'extraction, un ensemble de propriétés ou performances du circuit,
- l'analyse du statut de la simulation et de l'extraction: il faut distinguer une erreur de simulation, qui provoque un arrêt de la procédure, d'une erreur d'extraction due à une mauvaise définition des conditions de l'analyse (cas de l'extraction du *slew-rate* d'un op-amp pour laquelle la forme temporelle de l'impulsion d'entrée dépend de la rapidité du circuit),
- l'incrémentation d'une variable d'itération et la reprise de la simulation avec un nouveau jeu de paramètres, dans le cas où une erreur d'extraction a été détectée,
- l'exploitation du fichier de sortie du simulateur pour une mise en forme tabulaire,

- le lancement éventuel de post-processeurs (analyse des pôles et zéros par exemple) qui peuvent utiliser des résultats déjà acquis.

Une procédure ne nécessite généralement qu'une seule simulation, sauf dans les cas où les conditions de simulation dépendent des caractéristiques du circuit c'est à dire lorsque la plage correcte d'analyse du circuit est a priori inconnue [ACT92]. C'est par exemple le cas de la caractérisation du *slew-rate* d'un amplificateur par une simulation transitoire: le temps de simulation et les paramètres de l'impulsion d'entrée dépendent de la rapidité de l'amplificateur (cf Figure 47).

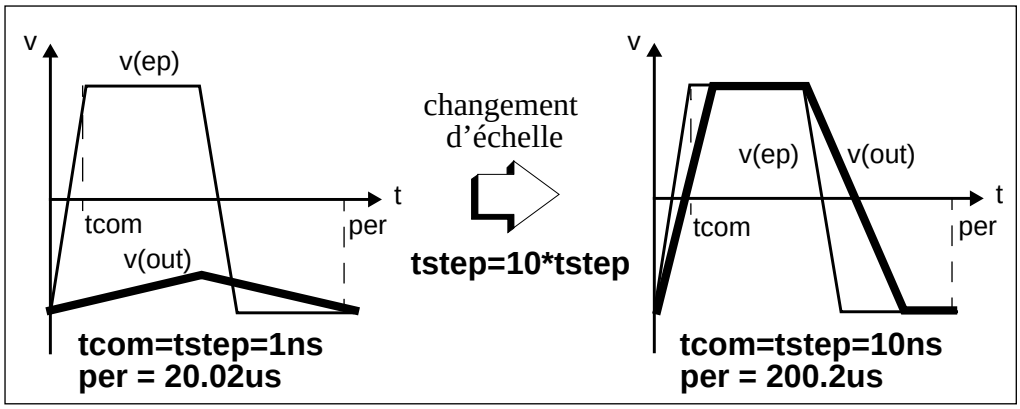


Figure 47 Réponse transitoire selon l'impulsion appliquée

L'outil résout ce problème par l'utilisation d'une procédure ***itérative*** sur un paramètre qui a été préalablement défini par l'utilisateur en tant que constante numérique. Ce dernier doit de plus spécifier la fonction d'incrémentation et le nombre maximal d'itérations. La boucle est alors exécutée jusqu'à ce que la mesure soit valide et en un nombre limite d'itérations. Par exemple, dans le cas de la mesure du slew-rate d'un op-amp décrite en Figure 44, on peut incrémenter le paramètre `tstep` qui définit l'unité de temps à partir duquel sont déterminés les paramètres de la commande d'analyse transitoire et la forme de l'impulsion d'entrée. Le changement d'échelle est par exemple réalisé par la fonction d'itération `tstep=10*tstep`.

D'autre part, certaines mesures nécessitent le lancement de programmes spécifiques, en plus du simulateur. C'est le cas du programme d'analyse des pôles et zéros associé à Eldo et qui doit être lancé après une simulation fréquentielle. Il délivre une liste de racines qui nécessite un filtrage pour éliminer les paires pôle-zéro qui se superposent. Un programme a été développé pour automatiser cette opération: pour contrôler la précision de la liste réduite, il

mesure à chaque étape la distance entre les courbes de gain et de phase qui sont issues de la simulation et celles calculées à partir de la fonction de transfert réduite, à des fréquences de contrôle qui sont spécifiées par l'utilisateur (par exemple fréquence de coupure, bande passante,...). Le paragraphe 5.3 donne plus de détails à ce sujet.

4 Utilisation des procédures de mesure

Les procédures de mesure, qui ont été stockées dans une bibliothèque après validation, sont exécutées soit pour la génération des paramètres d'un macro-modèle soit pour la simple création de *data-sheet*.

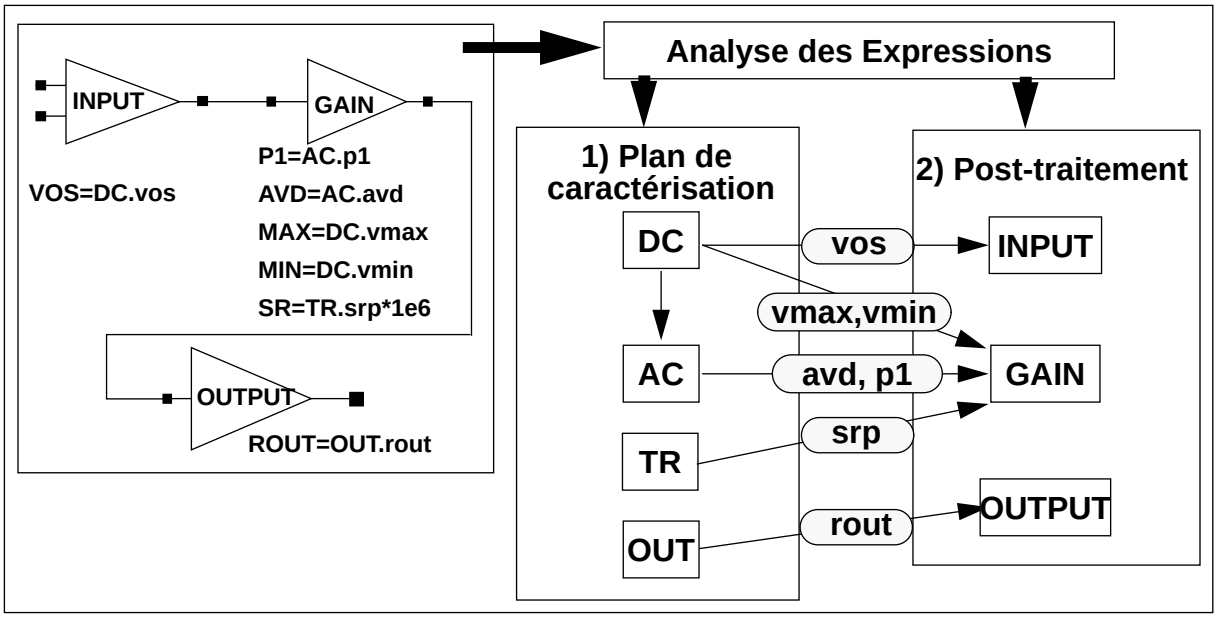


Figure 48 Première entrée: génération des paramètres d'un macro-modèle d'op-amp

4.1 Calcul des paramètres d'un macro-modèle

L'entrée de l'outil est dans ce cas un schéma constitué de blocs appartenant à une bibliothèque fonctionnelle associée à l'environnement. Chaque bloc est caractérisé par une liste de paramètres, dont les valeurs peuvent être définies par l'utilisateur comme des expressions faisant référence à des mesures disponibles dans la bibliothèque de caractérisation (cf Figure 43). Le *langage de spécification de mesures*, défini au paragraphe 3.3 pour la définition des paramètres de procédures, est ici utilisé. Par exemple, on peut définir le paramètre AVD définissant le gain différentiel d'un étage de gain comme la valeur moyenne du

gain déterminé par une procédure nommée `AC` exécutée dans les deux configurations `T1` et `T2`, selon l'expression: `AVD = (AC@T1.avd + AC@T2.avd)/2.`

L'analyse des expressions permet à l'environnement d'élaborer les commandes requises pour chaque bloc tant pour la caractérisation que pour le post-traitement des mesures afin d'évaluer leurs valeurs numériques (cf Figure 48). L'environnement offre de plus la possibilité de mettre à jour localement les conditions de mesure des différentes procédures requises, même si une procédure est implicitement appelée par une autre.

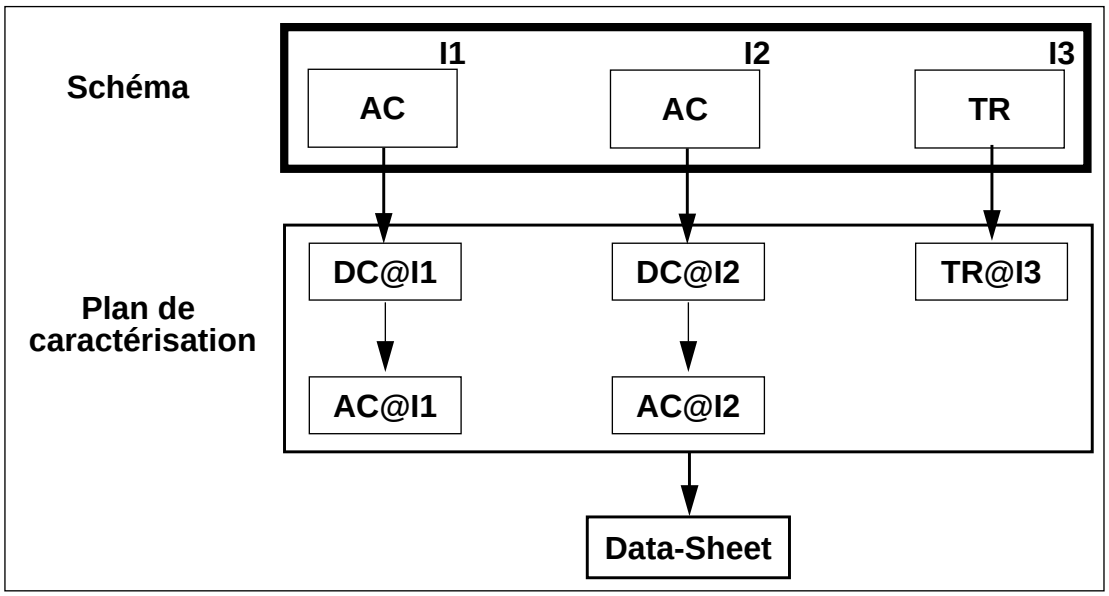


Figure 49 Seconde entrée: génération de *datasheet*

4.2 Génération de data-sheet

Après avoir généré le code du macro-modèle, il est indispensable d'en vérifier le comportement et les performances. Pour générer un simple *data-sheet*, la seconde entrée du schéma synoptique de la Figure 43 est utilisée. Il s'agit d'une ***entrée graphique***, basée sur l'éditeur de schéma de l'environnement de conception (cf Figure 49). Une procédure peut en effet être vue comme un composant, à la seule différence que les connexions sont établies à partir des expressions des conditions de mesure. Chaque procédure est identifiée par un nom donné par l'éditeur de schéma (nom d'appel ou d'*instance*) et qui est considéré par l'outil comme son nom de configuration.

Ainsi, il est possible de faire appel à deux procédures de même nom mais distinctes de par leur configuration (dans l'exemple de la Figure 49, AC@I1 et AC@I2 sont deux configurations de la procédure AC). Le programme de caractérisation tient aussi compte des

procédures qui sont implicitement définies dans les expressions des paramètres des procédures présentes sur le schéma. Ainsi dans le cas de la Figure 49, AC dépend de DC. Comme un nom de configuration doit être donné à chaque procédure, les procédures appelées prennent le même nom de configuration que les procédures appelantes: AC@I1 dépend de DC@I1 et AC@I2 dépend de DC@I2.

Enfin, le système permet à l'utilisateur de modifier localement les valeurs des paramètres, même si la procédure est implicitement définie.

5 La structure du système de caractérisation

5.1 Analyse des expressions des procédures et des blocs

Le coeur de l'outil de caractérisation est constitué du programme C nommé ***genawk***, qui analyse les expressions des paramètres des procédures et des blocs fonctionnels de macro-modèles. À partir de ces expressions, il génère de nombreux fichiers qui sont utilisés ultérieurement par des outils standards *Unix* tels que les programmes ***sed***, ***awk*** et ***makefile***, et par l'interface utilisateur elle-même:

- la mise à jour du fichier de commande du simulateur est réalisée par ***sed*** qui permet d'effectuer des substitutions de chaines de caractères dans un fichier,
- le calcul des valeurs numériques des paramètres des blocs fonctionnels ainsi que des procédures est effectuée par ***awk*** qui est capable de réaliser tout traitement de données et de générer des fichiers,
- le programme de caractérisation est en réalité un ***makefile***, qui est un programme d'ordonnancement de tâches, généralement utilisé pour la compilation de code. L'avantage du *makefile* est de gérer directement les dépendances d'une part entre les fichiers définissant les paramètres des étages fonctionnels (de suffixe *.aop*) et les fichiers de caractérisation (de suffixe *.res*), et d'autre part entre les fichiers de caractérisation eux-mêmes pour les procédures hiérarchiques.

5.1.1 Paramètres des procédures

La Figure 50 décrit l'ensemble des actions réalisées par ***genawk*** pour l'analyse des paramètres d'une procédure:

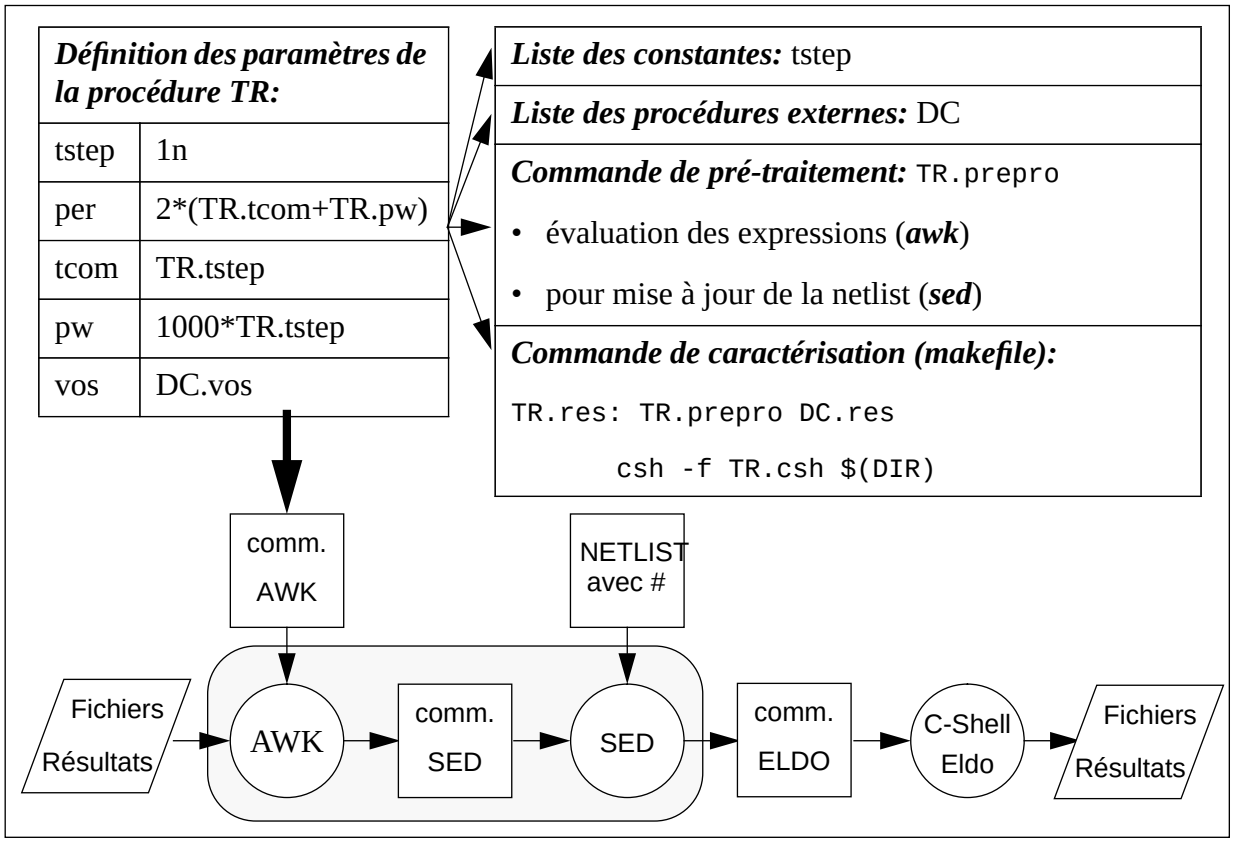


Figure 50 Analyse des paramètres d'une procédure

- ***définition de la liste des constantes***: les constantes numériques sont identifiées car elles peuvent être utilisées en tant que variables d'itération pour la définition d'une procédure itérative.
- ***définition de la liste des procédures externes***: cette liste est utilisée pour gérer les dépendances entre procédures. Elle permet aussi d'établir la liste complète des procédures nécessaires à un data-sheet ou à la génération des paramètres d'un macro-modèle.
- ***génération d'un programme de pré-traitement***: il s'agit en fait d'un fichier de commande *awk* pour le calcul numérique des expressions à partir de données se trouvant éventuellement dans plusieurs fichiers sources. Il génère aussi le fichier de commande du programme *sed* concernant la mise à jour du fichier de commande du simulateur Eldo, par substitution des paramètres précédés du caractère spécial # par leurs valeurs effectives.
  De plus, *genawk* doit analyser les relations de dépendance entre les divers paramètres afin de déterminer l'ordre de calcul. Ainsi, dans le cas de la Figure 50, le paramètre `per` devra être évalué après `tcom` et `pw`.
- ***génération de la commande de caractérisation*** de type ***makefile***: cette commande lance la

procédure de caractérisation.

5.1.2 Paramètres des blocs fonctionnels

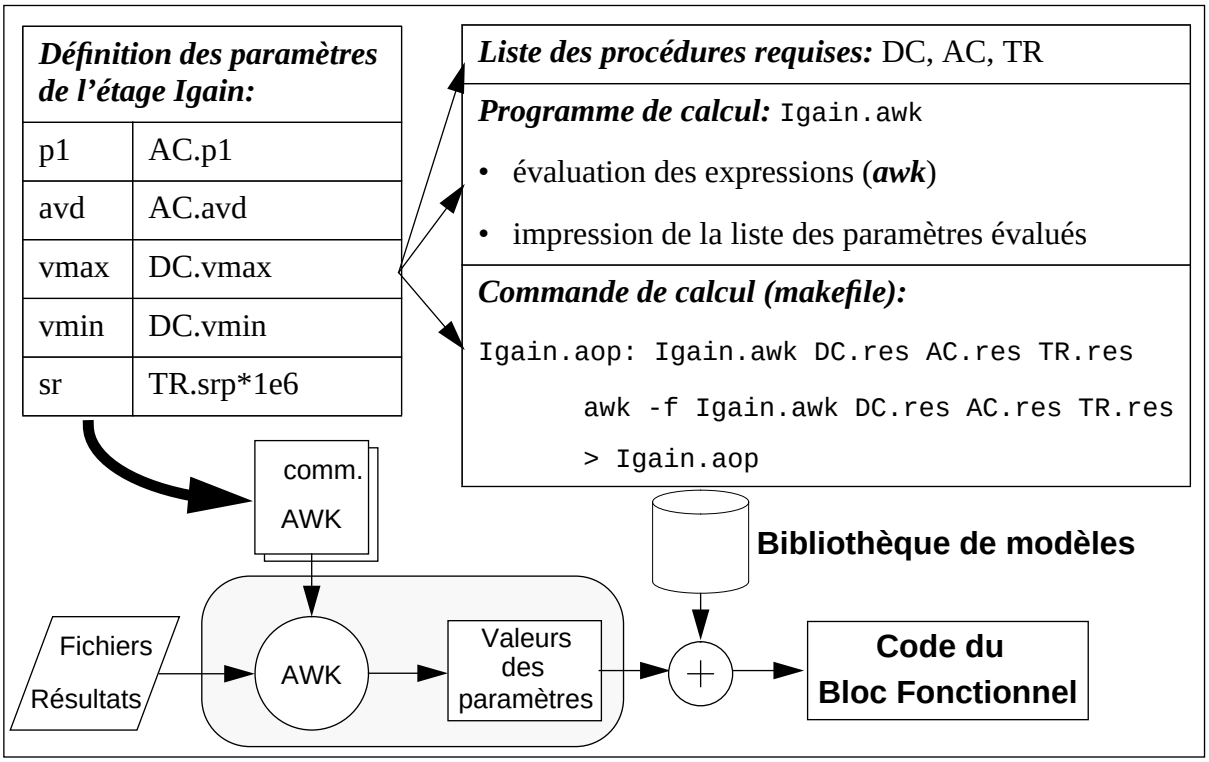


Figure 51 Analyse des paramètres d'un étage de macro-modèle

La Figure 51 décrit les actions réalisées par ***genawk*** lors de l'analyse des paramètres des étages fonctionnels d'un macro-modèle:

- ***définition de la liste des procédures requises***: cette liste définit les relations entre étages et procédures. Elle sert aussi à établir la liste complète des procédures nécessaires à la génération des paramètres d'un macro-modèle.
- ***génération d'un programme de pré-traitement***: il s'agit d'un fichier de commande *awk* qui effectue le calcul numérique des paramètres à partir d'un data-sheet préalablement généré. Il imprime enfin une liste définissant les valeurs des paramètres du bloc, qui est utilisée ultérieurement pour la génération du code du macro-modèle.
- ***génération de la commande de calcul,*** de type *makefile*: cette commande lance le programme *awk* de calcul des paramètres de l'étage considéré.

5.2 Exemple de caractérisation

5.2.1 Commande de pré-traitement

Voici le fichier de commande *awk*, de suffixe `.prepro`, qui correspond à la définition donnée en Figure 50 pour les paramètres de la procédure TR (vos n'a été ajouté que pour exemple):

```
$1 == “vos” { vos = $3 }
END{
   tstep = 1*1.0e-09
   tcom = tstep
   pw = 1000*tstep
   per = 2*(tcom+pw)
   vos = vos
   printf “s/#tstep/%g/g\n”,tstep
   printf “s/#tcom/%g/g\n”,tcom
   printf “s/#pw/%g/g\n”,pw
   printf “s/#per/%g/g\n”,per
   printf “s/#vos/%g/g\n”,vos
   printf “\n”
}
```

On vérifiera que l'ordre de calcul des paramètres temporels de la source impulsionnelle d'entrée (`tcom`, `pw`, et `per`) a été correctement établi par *genawk*.

Ce fichier permet de génèrer, lors de l'exécution de la procédure, le fichier de commande *sed* suivant (en supposant que l'offset `vos`, déterminé par DC, est égal à 1e-6 V):

```
s/#tstep/1e-09/g
s/#tcom/1e-09/g
s/#pw/1e-06/g
s/#per/2.002e-06/g
s/#vos/1e-06/g
```

5.2.2 Commande de post-traitement

Cet exemple concerne la procédure TR, qui permet de déterminer les temps d'établissement, les slew-rates et les dépassements pour un front d'entrée montant puis descendant. Le fichier de commande *awk*, qui est automatiquement généré lors de la définition des fonctions d'extraction de la procédure, est le suivant:

```
BEGIN{ printf ";----------------------\n"
       printf ";Test   : TR\n"
       printf ";Date   : %s\n",date
       printf ";----------------------\n"
}
```

```
$1=="*" && $2~/\$SETP/ {printf "SETP = %g\n",$4}
$1=="*" && $2~/\$SETN/ {printf "SETN = %g\n",$4}
$1=="*" && $2~/\$SRP/ {printf "SRP = %g\n",$4}
$1=="*" && $2~/\$SRN/ {printf "SRN = %g\n",$4}
$1=="*" && $2~/\$OVERN/ {printf "OVERN = %g\n",$4}
$1=="*" && $2~/\$OVERP/ {printf "OVERP = %g\n",$4}
```

5.2.3 Procédure de caractérisation

Le code d'une procédure est automatiquement généré par l'interface, à la fin de la phase d'élaboration et sous la forme d'un programme *Unix* ***C-Shell*** (suffixe `.csh`). Il exécute les actions définies en Figure 46. Deux structures différentes sont employées selon que la procédure est itérative ou non. Dans le cas d'une procédure itérative, une boucle *while* est ajoutée ainsi que des commandes relatives à l'incrémentation de la variable de boucle. Le programme d'une procédure transitoire est définie de la manière suivante:

```
#!/bin/csh
set ELDO=/...
set TSTEP=1e-8
@ ERROR=1       # indicateur d'erreur
@ SIMSTEP=1     # compteur
# Modification du pré-processeur car TSTEP est mis à jour dans la boucle
sed '/^TSTEP/d' TR.prepro > /tmp/TR.prepro
mv /tmp/TR.prepro TR.prepro
# Boucle:
while( ($ERROR == 1) && ($SIMSTEP <= 2))
  echo mac: simulation step ${SIMSTEP}:
  # Pré-processeur
  nawk -f TR.prepro TSTEP=$TSTEP  > TR.sed
  sed -f TR.sed TR.inp > TR.cir
  # Lancement de la simulation à travers un autre script
  $ELDO TR.cir
  # Traitement du statut de la simulation et extraction
  switch ($status)
       case 0: # Pas d'erreur
            @ ERROR=0; breaksw
       case 1,2: # Erreurs de simulation
            exit 1
       case 3: # Erreur d'extraction
            @ ERROR=1; breaksw
  endsw
  # Mise à jour de TSTEP
  set TSTEP=`nawk 'BEGIN { print TSTEP*10 }' TSTEP=$TSTEP `
  @ SIMSTEP+=1
end
# Post-Traitement: création du Data-Sheet
```

```
set DATE=`date '+%d/%m/%y/%H:%M'`
nawk -f TR.postpro date=$DATE TR.chi > TR.res
if ($NOERROR == 1) exit 0
exit 1
```

5.2.4 Calcul des paramètres d'un bloc

Le fichier de commande awk, de suffixe `.awk`, permettant de déterminer les paramètres de l'étage de gain en fonction des résultats des caractérisations DC, AC et TR (cf Figure 48), est le suivant:

```
BEGIN{  printf "UmacParamList=\" " }
$1 == "srp" { srp = $3 }
$1 == "vmin" { vmin = $3 }
$1 == "vmax" { vmax = $3 }
$1 == "avd" { avd = $3 }
$1 == "p1" { p1 = $3 }
END{
        p1 = p1
        avd = avd
        vmax = vmax
        vmin = vmin
        sr = srp*1e6
        printf "p1=%g ",p1
        printf "avd=%g ",avd
        printf "vmax=%g ",vmax
        printf "vmin=%g ",vmin
        printf "sr=%g ",sr
        printf "\"\n"
}
```

où `UmacParamList` est une liste *Skill* utilisée par l'environnement de conception pour la génération du code du macro-modèle et dont la valeur, définie dans le fichier `Igain.aop`, pourrait être par exemple:

```
UmacParamList="p1=100 avd=100 vmax=14.5 vmin=-14.5 sr=10.0"
```

5.2.5 Plan de caractérisation

Voici un exemple simplifié correspondant à la caractérisation des paramètres du macro-modèle défini à la Figure 48 comportant un étage d'entrée (Iinput), de gain (Igain) et de sortie (Ioutput):

```
.IGNORE:
.SUFFIXES:
.SUFFIXES: .prepro .res .awk .aop
# Première partie:
```

```
# génération du datasheet nommé macromodel.res
TESTLIST=DC AC TR OUT
$(TESTLIST):  $$@.res
macromodel.res:
   rm -f  macromodel.res
   for file in $(TESTLIST); do \
        cat  $$file.res >>  macromodel.res; \
        done
DC.res: DC.prepro
   DC.csh
AC.res: AC.prepro DC.res
   AC.csh
TR.res: TR.prepro DC.res
   TR.csh
OUT.res: OUT.prepro
   OUT.csh

# Seconde partie:
# calcul des paramètres des étages Iinput, Igain, Ioutput
STAGELIST=Iinput Igain Ioutput
$(STAGELIST):  $$@.aop
all: $(STAGELIST)  macromodel.res

Iinput.aop: Iinput.awk DC.res
   awk -f Iinput.awk DC.res >  Iinput.aop
Igain.aop: Igain.awk DC.res AC.res TR.res
   awk -f Igain.awk DC.res AC.res TR.res >  Igain.aop
Ioutput.aop: Ioutput.awk OUT.res
   awk -f Ioutput.awk OUT.res > Ioutput.aop
```

La première partie correspond aux commandes de caractérisation mettant en jeu la liste de procédures `DC AC TR OUT`. Toutes les commandes élémentaires de type *makefile* qui ont été générées pour chacun des blocs sont regroupées dans la seconde partie du *makefile*. Ce programme est lancé par la commande **“make all”**. Dans le cas d’une simple génération de datasheet, seule la première partie serait présente et le programme serait lancé par **“make macromodel.res”**. On note que l’étage `Igain` dépend des résultats de la procédure `AC` qui elle-même dépend de ceux de `DC`. Le *makefile* exécute donc successivement `DC` puis `AC` avant de générer le fichier `Igain.aop` définissant les valeurs des paramètres du bloc de gain.

5.2.6 Exemple de data-sheet

```
;----------------------
;Test: opa_vos
;Date: 06/10/94/15:46
;----------------------
VOS = 4.53292e-06
;----------------------
;Test: opa_tr
;Date: 06/10/94/15:47
;----------------------
SRN = -3.33785e+07
SRP = 3.56246e+07
```

```
;----------------------
;Test: opa_ac
;Date: 06/10/94/15:47
;----------------------
UGB = 6.83509e+07
AD = 98.0202
CMRR = 123.758
P1 = 1341
PIm1 = 6.38126e+07
PIz1 = 0.749927
P2 = 7.45442e+07
ZIm1 = 8.93822e+07
ZIz1 = 0.838981
Z1 = 3.0527e+08
Z2 = 3.55672e+08
```

5.3 Post-traitement des pôles et zéros

Dans Eldo, une commande placée dans la netlist `.pz v(<noeud>)` provoque la création d'un fichier de données qui est analysé par un outil nommé `pz` à lancer après la simulation. Cet outil comporte une première phase de calcul des pôles et zéros et une seconde phase de tri et d'élimination des paires de pôle-zéro qui se superposent. Pour une caractérisation automatique, un programme supplémentaire a été développé qui se substitue au filtrage interactif de `pz`. Il est basé sur le même principe, à savoir qu'un pôle P et un zéro Z sont éliminés si les conditions suivantes sont vérifiées (cf Figure 52a):

$$|Re(P) - Re(Z)| < TH \times |Re(P)| + TH \quad \text{et} \quad |Im(P) - Im(Z)| < TH \times |Im(P)| + TH .$$

où TH est un seuil dont la valeur est strictement comprise entre 0 et 1. Notons que le deuxième terme de chaque membre de droite est généralement négligeable, sauf si P est réel ou imaginaire pur ou proche de 0. Notons que les racines d'une fonction de transfert complexe sont soit réelles soit complexes conjuguées. Ainsi, si p et z vérifient les conditions précédentes, quatre cas peuvent se présenter:

- p et z sont réels: on élimine p et z.
- p et z sont complexes non réels: comme il existe dans la liste les quantités conjuguées p et z, on peut éliminer p, $\overline{p}$, z et $\overline{z}$.
- p est complexe non réel et z est réel: la partie imaginaire de p peut être négligée; on élimine

z et $\bar{p}$ et on annulle la partie imaginaire de p.

- z est complexe non réel et p est réel: de même que précédemment, on élimine p et $\bar{z}$ et on annulle la partie imaginaire de z.

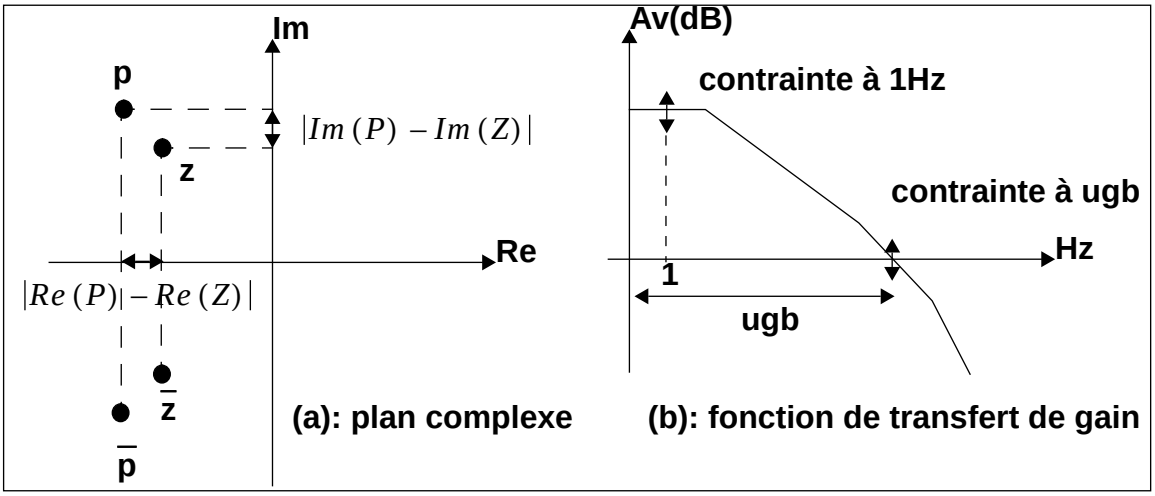


Figure 52 Filtrage des racines de la fonction de transfert

Le filtrage réalisé est d'autant plus efficace que *TH* est élevé, au détriment cependant de la précision. C'est pourquoi une nouvelle procédure de vérification a été élaborée afin de contrôler l'écart entre les courbes réelles (du simulateur) et celles obtenues par calcul à partir de la fonction de transfert réduite, et ce pour chaque valeur de *TH* et à certaines fréquences. L'utilisateur doit donc définir, au moment de l'élaboration de la procédure, une ***liste de contraintes sur le gain et sur la phase***, à des fréquences données. Si toutes les contraintes sont vérifiées, *TH* est incrémenté de 0.1 jusqu'à 0.9 et tant que la somme des nombres de pôles et zéros reste non nulle. Par exemple, l'utilisateur peut spécifier que la fonction de transfert réduite donne pour la bande-passante, nommée ***ugb***, une valeur de gain proche du circuit réel à 1dB près, la donnée ***ugb*** étant calculée par une fonction d'extraction ou ***macro*** préalablement définie par l'utilisateur (cf Figure 52b).

Enfin, ce nouveau post-processeur possède une sortie graphique qui permet de comparer la réponse de la fonction de transfert réduite avec celle du circuit étudié.

6 Interface utilisateur

Le but de ce paragraphe est de donner quelques exemples des éléments de l'interface utilisateur de l'environnement de caractérisation, concernant en particulier la phase d'élaboration de nouvelles procédures de mesure, lors de laquelle l'utilisateur doit construire

les fonctions d'extraction, donner les valeurs par défaut des paramètres de la procédure et enfin choisir le mode d'exécution.

6.1 Définition des fonctions d'extraction

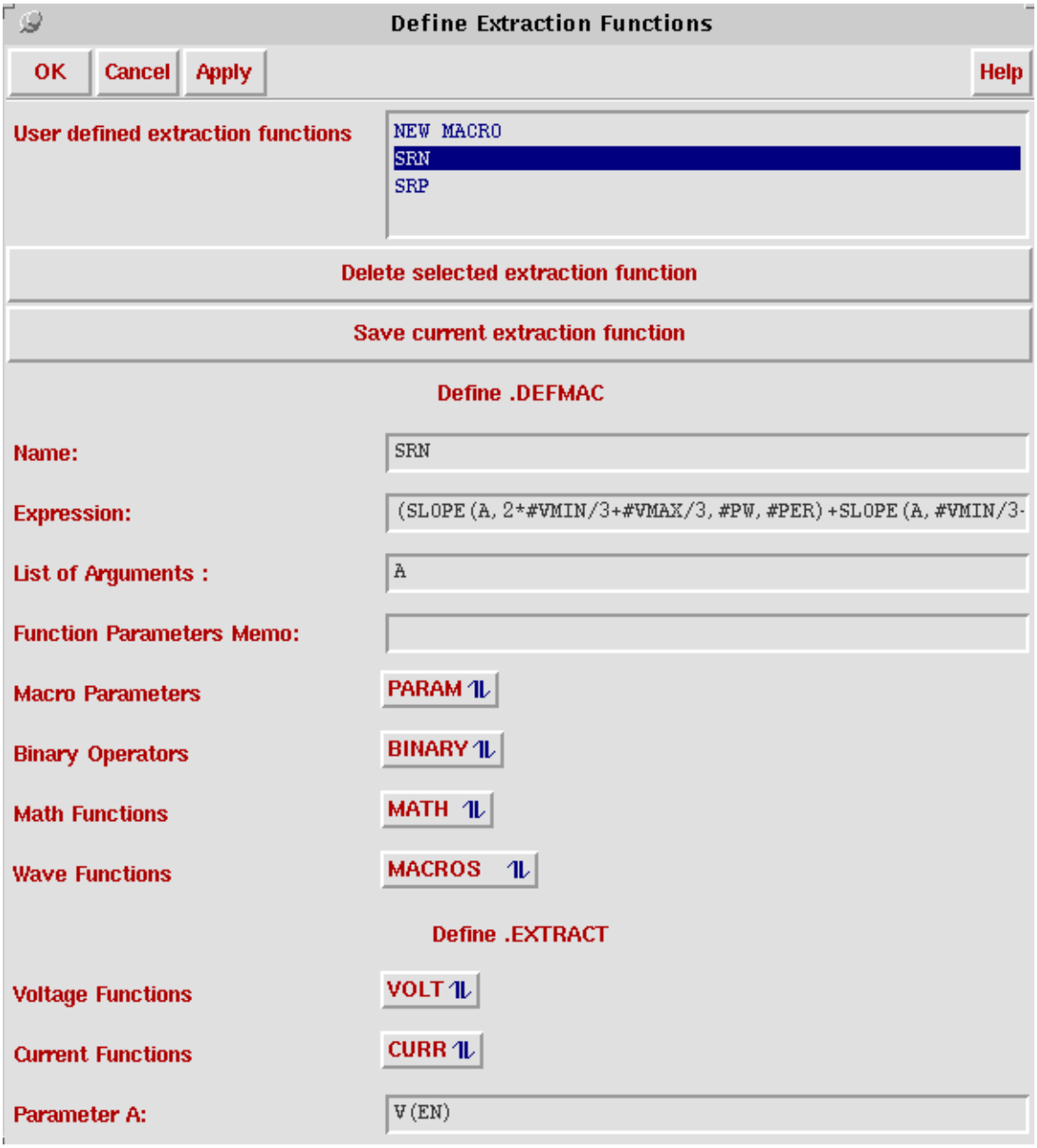


Figure 53 Définition des macro-fonctions d'extraction

Pour créer une nouvelle macro-fonction (cf Figure 53), l'utilisateur doit cliquer avec la souris sur le champ "NEW MACRO" de la fenêtre intitulée "User defined extraction functions" qui donne la liste de toutes les fonctions créées. Les champs suivants doivent alors

être renseignés:

- le nom, par exemple SRN qui désigne le slew-rate négatif,
- l'expression de définition, en utilisant les menus déroulants "Macro Parameters", "Binary Operators", "Math Functions" et "Wave Functions". Le mode d'utilisation des fonctions traitant les courbes est explicité (champ "Memo").
- la liste "abstraite" des arguments de la macro-fonction (par exemple A). Ces noms sont ceux utilisés dans la définition de l'expression. Leur valeur est ensuite précisée dans des champs créés dynamiquement ("Parameter A:"). Si la valeur est un nom de noeud, il suffit de cliquer sur le schéma (idem pour une source de tension à travers de laquelle est mesurée un courant).
- le bouton "Save current extraction function" permet de placer la macro-fonction dans la liste des "User defined extraction functions", alors que "Apply" et "OK" effectuent une sauvegarde définitive.

6.2 Définition des valeurs des paramètres de la procédure

La liste des paramètres, déterminée par le système après analyse de la description du schéma (*netlist*), des commandes, des fonctions d'extraction,... est affichée dans un ordre quelconque (cf Figure 54). Rappelons qu'un paramètre de procédure est définie par #NOM. Un langage particulier permet de définir des relations entre les paramètres et même avec des résultats d'autres procédures. Le système élabore à partir de ces expressions un certain nombre de fichiers et en particulier la commande de caractérisation adéquate et le programme de mise à jour du fichier de commande du simulateur.

6.3 Préparation de la simulation

L'utilisateur peut définir une procédure itérative, dite 'MultiStep' (cf Figure 55) qui effectue des simulations jusqu'à ce que l'extraction soit correcte. Les champs correspondants sont:

- 'Incremental Parameter': l'utilisateur doit préciser le nom de la variable qui sera incrémentée à partir de la liste présentée par ce champ cyclique. Dans la Figure 55, on a choisi la variable TSTEP à partir de laquelle tous les paramètres temporels ont été définis (cf

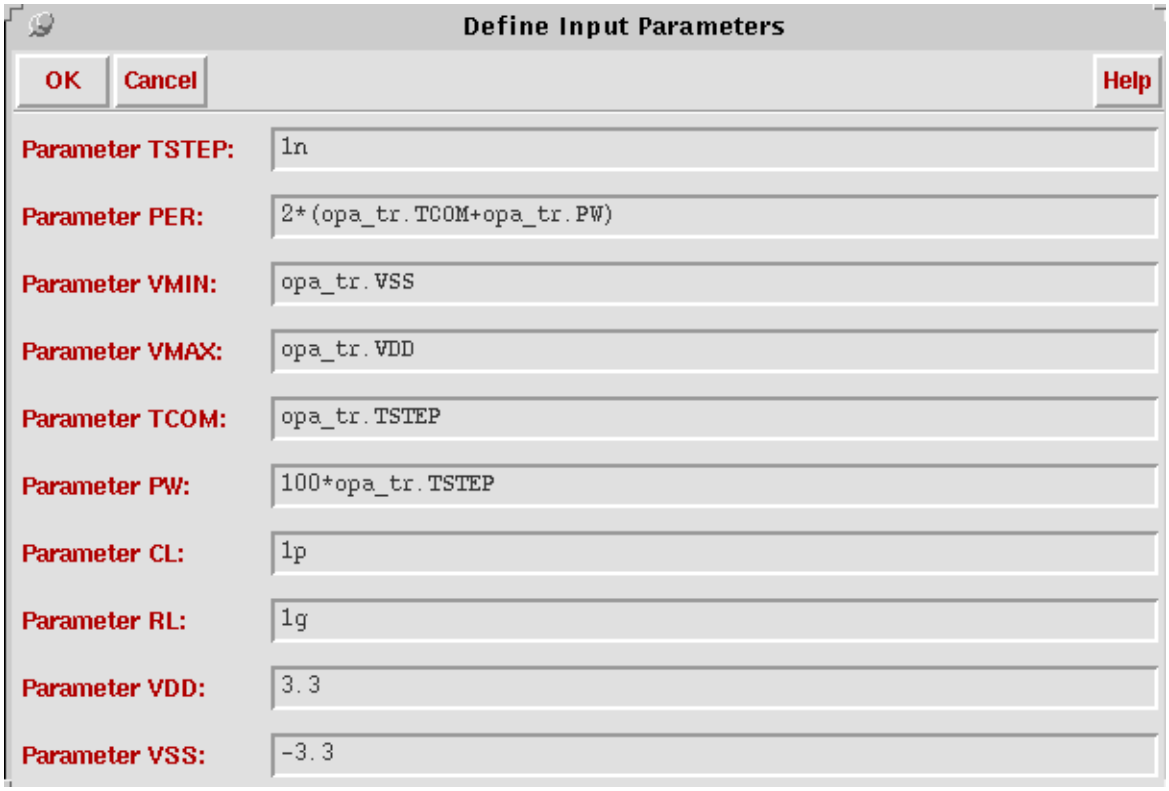


Figure 54 Définition des paramètres de la procédure *opa_tr*

Figure 54).

- 'Initial value': valeur initiale de la variable précédemment définie,
- 'Incremental function': par exemple 10*TSTEP,
- 'Number of steps': cela correspond au nombre maximal d'itérations.

L'utilisateur peut aussi faire appel à un post-processeur d'analyse de pôles et zéros (cf Figure 56), auquel il faut préciser le nom du noeud de sortie 'Node'. Un certain nombre de paramètres concernent le filtrage automatique de la liste des pôles et zéros rendue par le programme:

- 'Initial threshold': valeur initiale, entre 0 et 1, du seuil qui permet d'identifier les paires pôle-zéro qui se superposent,
- 'Maximum frequency': seules les racines en dessous de cette fréquence sont considérées,
- 'Freq. Gain Tol. Values': pour garder une bonne précision, l'utilisateur doit spécifier une liste de contraintes qui sont entrées sous la forme de triplets fréquence-gain-tolérance, tels que: 1 AD 1 ; UGB 0 1, où AD et UGB sont des macro-fonctions, présentées dans le champ cyclique 'Extraction Functions' et préalablement définies par l'utilisateur. Elles rendent la

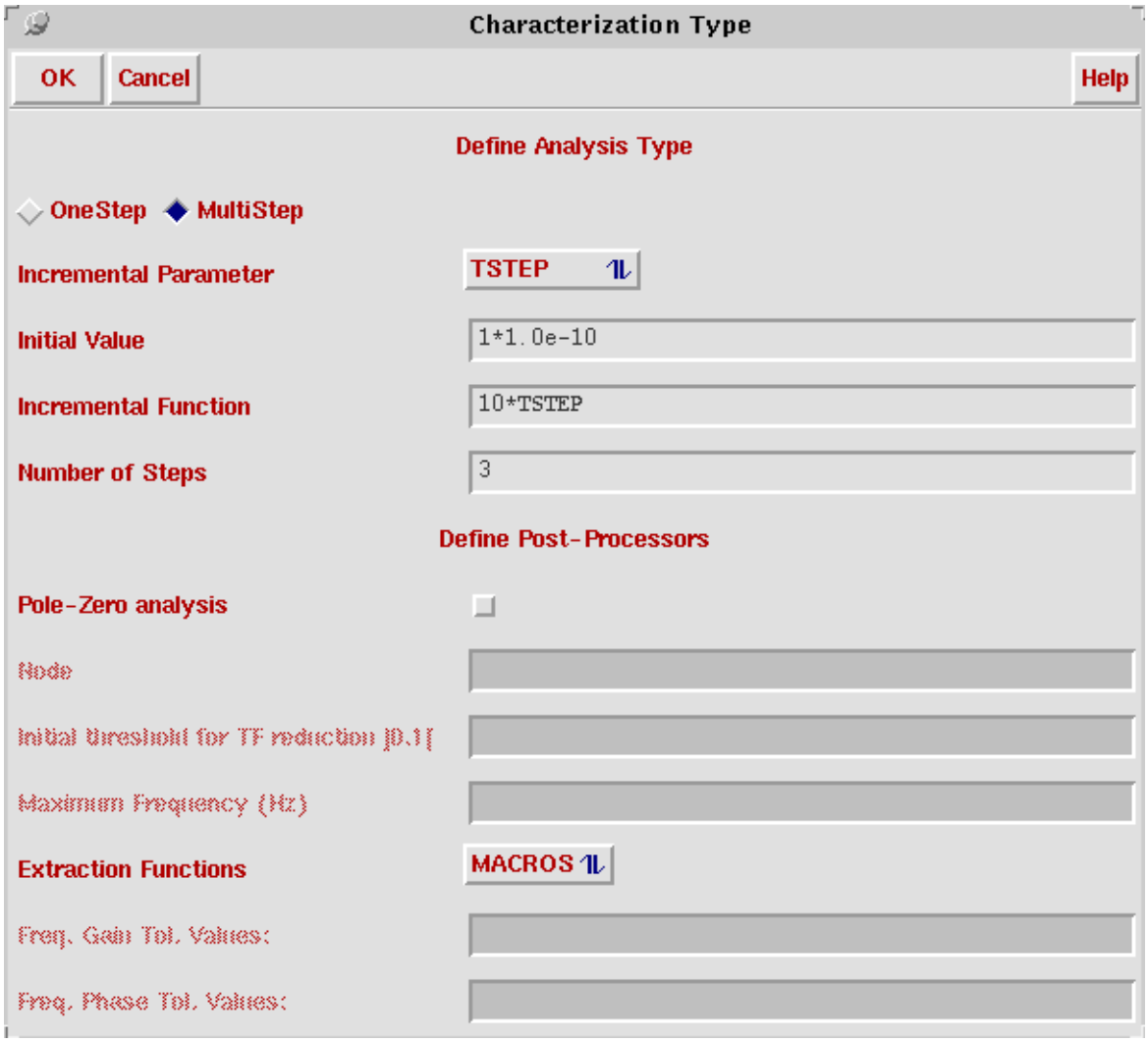


Figure 55 Définition du type de procédure

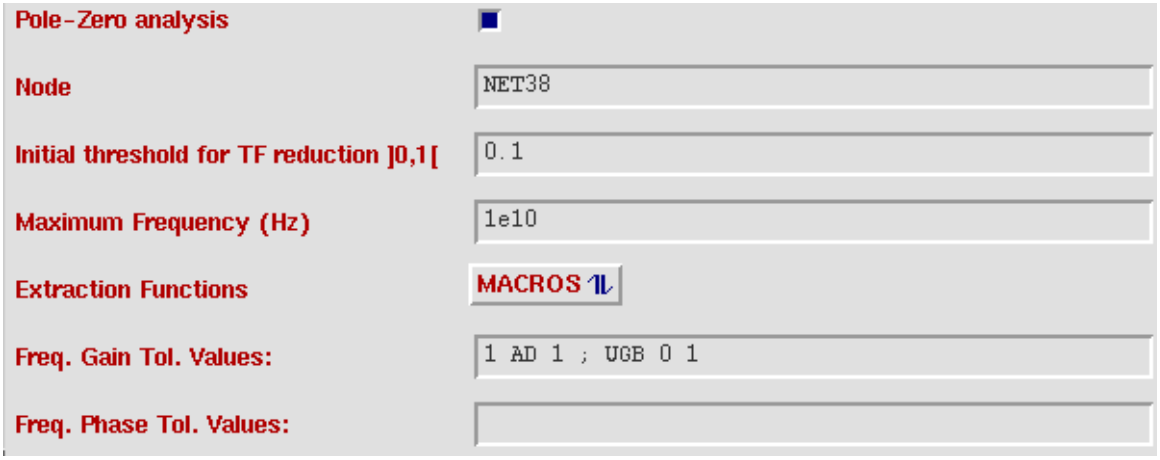


Figure 56 Définition du Pole-Zero post-processeur

valeur du gain respectivement à 1 Hz et à la bande-passante.

- ‘Freq. Phase Tol. Values’: on peut aussi entrer une liste de contraintes sur la phase sous la

forme de triplets fréquence-phase-tolérance.

6.4 Sauvegarde des données de la procédure

Certaines données qui ont été générées lors de l'élaboration de la procédure sont stockées dans une bibliothèque *Cadence* sous forme de *vues* associées au schéma du *test-bench* et à son symbole graphique (cf Figure 57). La fonction des différentes données est décrite par le Tableau 13.

Tableau 13 Définition des données stockées dans la bibliothèque pour une procédure

Nom de la vue	Fonction
UNImacCONFIGtext	fichier *Skill* mémorisant une configuration à partir de laquelle on peut construire une procédure similaire
UNImacEXTRACTtext	description des macro-fonctions (commandes *.extract* et *.macro* de Eldo)
UNImacMAKEtext	ligne de commande *makefile* qui est intégrée au programme de caractérisation
UNImacNETLISTtext	fichier de commande Eldo paramétré (*#param*)
UNImacPARAMtext	définition des paramètres de la procédure
UNImacPOSTPROtext	programme *awk* de post-traitement du fichier de sortie du simulateur Eldo pour une mise en forme tabulaire
UNImacPREPROtext	programme *awk* de calcul des paramètres de configuration de la procédure
UNImacSCRIPTtext	programme *C-Shell* de la procédure

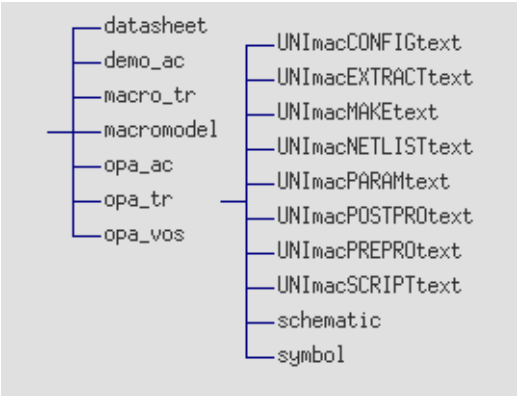


Figure 57 Vues de la bibliothèque correspondant à la procédure *opa_tr*

7 Bibliothèque des procédures de mesure

Une bibliothèque de procédures de mesure a été développée pour la caractérisation d'op-amps et a été utilisée pour la macro-modélisation du MC33272, présentée au Chapitre 5. Le Tableau 14 définit les principaux paramètres des procédures et les performances extraites dans chaque cas. Les schémas sont décrits en Annexe 6.

Tableau 14 Procédures de mesure pour un op-amp

Procédure	Analyse	Paramètres du schéma	Performances Mesurées
VOS_test	OP	alimentations, T	tension de décalage, courants d'entrée
DC_test	DC	alimentations, T, analyse (dcmin, dcmax, dcstep)	tensions maxi. & mini., gain, *swing*, tension de décalage
TR_test	Transitoire	alimentations, T, impulsion d'entrée (tcom, per,...) & analyse (tstep, tstop)	slew-rates positifs et négatifs, dépassements, temps d'établissement
AC_test	AC	alimentations, T, analyse (fmax)	gain différentiel, CMRR, bande passante, marges de gain & phase, pôles & zéros
ICC_test	DC param	alimentations, T, analyse (imin, imax, istep)	courants de sortie maxi. et mini.
ROUT_test	AC	alimentations, T, analyse (fmax)	résistance DC, capacité équivalente
SRR_test	AC	alimentations, T, analyse (fmax)	facteurs de réjection d'alimentations

8 Conclusion

Ce système de caractérisation de cellules analogiques qui est couplé au calcul des paramètres d'un macro-modèle a été utilisé avec succès pour le développement d'un modèle d'op-amp de SGS-Thomson et étend les fonctionnalités de l'environnement analogique actuel de la compagnie [MLN93]. Grâce à cet outil et à la bibliothèque de macro-modèles qui lui est associée, le gain de temps pour le développement de macro-modèles devrait être appréciable,

bien qu'il n'ait pas encore pu être quantifié par les concepteurs eux-mêmes.

Concernant le module de caractérisation, des évolutions sont envisagées comme l'utilisation du langage *Simpilot* [Sim94] permettant de commander des simulations Eldo en définissant les valeurs des conditions d'analyse (commandes, stimuli, éléments de charge) et aussi de traiter les courbes de manière très efficace. Le calcul des paramètres des modèles comportementaux pourrait être réalisé par ce même programme.
La caractérisation des *dérives statistiques des performances*, dues aux fluctuations technologiques, pourrait être de plus un atout important. Dans ce but, on peut envisager l'utilisation du programme *Aspire* [Asp95], associé à *Simpilot*, et qui permet de générer, par la méthode *RSM*, des polynômes des performances en fonction de paramètres de composants.
Enfin, pour la détermination des pôles et zéros, une approche d'approximation du type AWE (cf Chapitre 2) mériterait d'être étudiée pour remplacer la méthode délicate d'analyse et de filtrage actuellement suivie.

Concernant l'aspect du calcul des paramètres de modèle, il serait utile de disposer d'un module d'optimisation pour la définition de fonctions non-linéaires, généralement décrites par des tables de valeurs. L'utilisation du logiciel d'optimisation *Opsim* [Ops94], associé à *Simpilot*, pourrait être envisagée.

Chapitre 7: Traduction des langages de modélisation

1 Introduction

Comme nous l'avons déjà vu au Chapitre 3, il existe actuellement sur le marché un certain nombre de langages de modélisation comportementale analogique, mais aucun n'est pour l'instant accepté en tant que standard. Or cette absence freine considérablement l'échange de modèles entre les compagnies industrielles. C'est pourquoi il a été décidé d'étudier la possibilité de traduction d'un langage à un autre, sans perte de précision quant à la description.

Au cours de cette thèse, plusieurs langages ont été abordés, ***FAS, CFAS, HDL-A*** et ***MAST***. Leurs propriétés générales sont décrites dans le Tableau 15.

Tableau 15 Propriétés des langages étudiés

Langage	FAS	C-FAS	HDL-A	MAST
Simulateur	ELDO	ELDO	ELDO	SABER
Analogique	oui	oui	oui	oui
Digital	non	non	oui	oui
Comportemental	oui	oui	oui	oui
Structurel	non	non	oui	oui
Société	ANACAD			ANALOGY

Précisons dès à présent que le langage FAS constitue le langage source de tous les traducteurs étudiés. C'est en effet le langage initialement adopté par SGS-Thomson et avec lequel une bibliothèque de modèles fonctionnels a tout d'abord été développée. C'est un langage de haut-niveau qui est cependant limité aux descriptions comportementales analogiques.

Aujourd'hui, ce langage est devenu obsolète au sein de la société et tend à être remplacé par HDL-A, langage qui devrait être assez proche du futur standard VHDL-A ou IEEE 1076.1. D'autre part, HDL-A présente de sérieuses améliorations par rapport à FAS, surtout en ce qui

concerne la mise au point des modèles, par l'intermédiaire d'un "débogueur". Le traducteur ***FAS vers HDL-A*** a facilité le transfert de tous les modèles déjà développés dans le nouveau langage.

Le langage C-FAS correspond à l'interface C du langage FAS. Alors que FAS est un langage interprété, C-FAS doit être compilé et il permet, par conséquent, d'obtenir un gain assez important en temps de simulation. La traduction ***FAS vers CFAS*** a été étudiée dans ce but et tous les modèles de la bibliothèque fonctionnelle peuvent maintenant être utilisés sous forme compilée C-FAS.

Enfin, MAST est un langage de modélisation comportementale analogique/digital qui est très utilisé dans l'industrie électro-mécanique du fait de la richesse de sa bibliothèque de composants multi-disciplinaires. Du fait de la complexité croissante des systèmes développés, les équipementiers souhaitent en vérifier le fonctionnement au plus haut-niveau. Ils ont donc besoin des modèles comportementaux des différents circuits que SGS-Thomson peut concevoir dans ce domaine. En l'absence de standard, le traducteur ***FAS vers MAST*** est une solution provisoire qui a permis de résoudre les problèmes d'échange de modèles entre la société et un de ses clients.

Cependant, des langages comme MAST et HDL-A évoluent constamment, FAS tendant à disparaître. Les traducteurs qui ont été développés ne sont donc valides que pour certaines versions et ne devraient avoir qu'une durée d'utilisation assez brève. Nous nous intéresserons donc surtout aux méthodologies de traduction et à la description des problèmes rencontrés, les caractéristiques des traducteurs FAS vers HDL-A, FAS vers CFAS et FAS vers MAST étant décrites respectivement en Annexe 7, Annexe 8 et Annexe 9.

2 Élaboration d'un traducteur

2.1 Méthodologie

Un programme de traduction de langage est similaire à la partie frontale d'un compilateur et comporte les étapes successives suivantes [ASU86]:

- analyse *lexicale*: elle consiste en la reconnaissance des unités lexicales ou lexèmes du langage source, c'est à dire les mots clés, les opérateurs, les identificateurs, les nombres entiers ou réels, etc...

- analyse *syntaxique* ou grammaticale: elle regroupe les lexèmes en structures grammaticales qui sont représentées par un arbre syntaxique (cf Figure 58),
- analyse *sémantique*: elle contrôle si le programme source comporte des erreurs sémantiques par exemple concernant le type de données,
- *génération du code* qui correspond ici à l'écriture du modèle dans le format du langage cible.

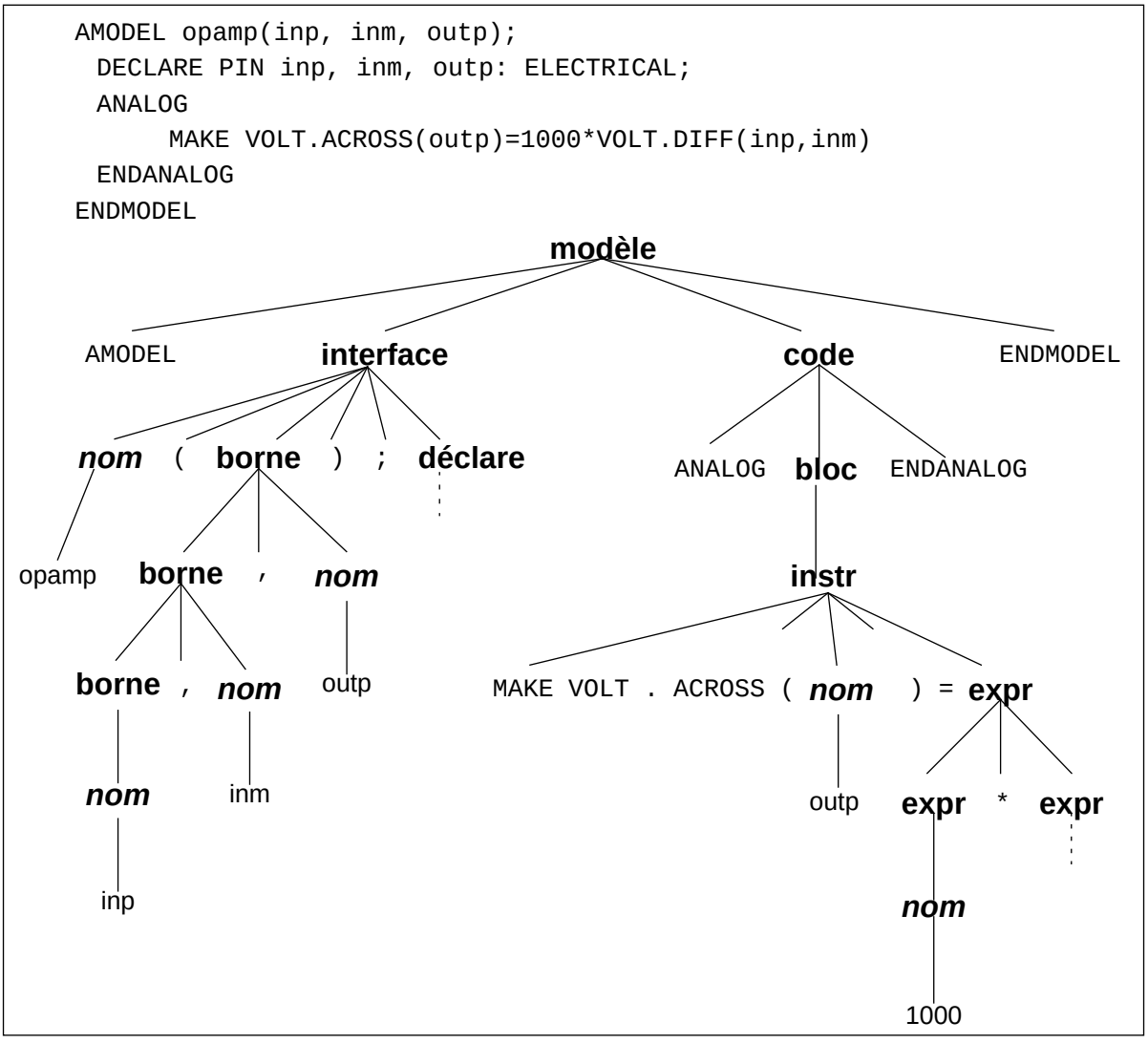


Figure 58 Arbre syntaxique partiel d'un modèle ELDO-FAS

La Figure 58 donne l'exemple d'un arbre syntaxique partiel d'un modèle FAS. Sa structure hiérarchique correspondrait à la spécification syntaxique suivante, exprimée dans le format BNF (acronyme de *Backus-Naur Form*), qui est couramment utilisé:

modèle ::= AMODEL **interface init code** ENDMODEL
interface ::= ***nom*** (**borne**) ; **déclare**
déclare ::= DECLARE PIN **borne** : ELECTRICAL ;

```
...
borne     ::=     nom
            |     borne , nom
code      ::=     ANALOG bloc ENDANALOG
bloc      ::=     instr
            |     bloc instr
instr     ::=     MAKE nom = expr
            |     MAKE VOLT . ACROSS ( nom ) = expr
...
expr      ::=     nombre
            |     nom
            |     VOLT . DIFF ( nom , nom )
            |     expr * expr
...
```

où:

- le signe **::=** désigne une règle syntaxique et la barre verticale | une alternative,
- AMODEL, ENDMODEL, INITIALIZE,... constituent les mots clés du langage. Avec les symboles **(),;** et l'opérateur multiplicatif, ils constituent les unités lexicales appelées symboles terminaux.
- ***nom*** et ***nombre*** sont des éléments non-terminaux qui sont composés d'éléments alphanumériques terminaux. Leur définition peut être donnée dans l'analyseur lexical.
- **modèle**, **interface**, **déclare**,... sont les noms des règles grammaticales, aussi appelées *productions*. Ce sont des éléments non-terminaux. Remarquons l'existence de règles à plusieurs alternatives (**instr, borne** ou **expr**).
  De plus, les productions **bloc** et **borne**, spécifiant des liste d'éléments, sont dites *récursives à gauche*: par exemple pour **borne**, la première règle, notée ***nom***, correspond seulement au premier élément et la seconde (**borne , *nom***) à tous les éléments suivants.
- **déclare** est en réalité une règle récursive, comme **bloc**, car d'autres variables sont à déclarer (paramètres, constantes, locales,...).

Enfin, l'analyseur syntaxique est à l'origine de la construction d'une structure de donnée, appelée *arbre abstrait*, à partir de laquelle sera généré le code de sortie. *Aho et al.* [ASU86] en donne la définition suivante: "un *arbre abstrait* est une forme *condensée* d'arbre syntaxique, adaptée à la représentation des constructions d'un langage".
Les opérateurs et les mot clés n'y apparaissent pas car ils sont en fait "inclus" dans le nom de l'*étiquette* donnée à chaque noeud qui peut être implémenté sous la forme d'une structure à

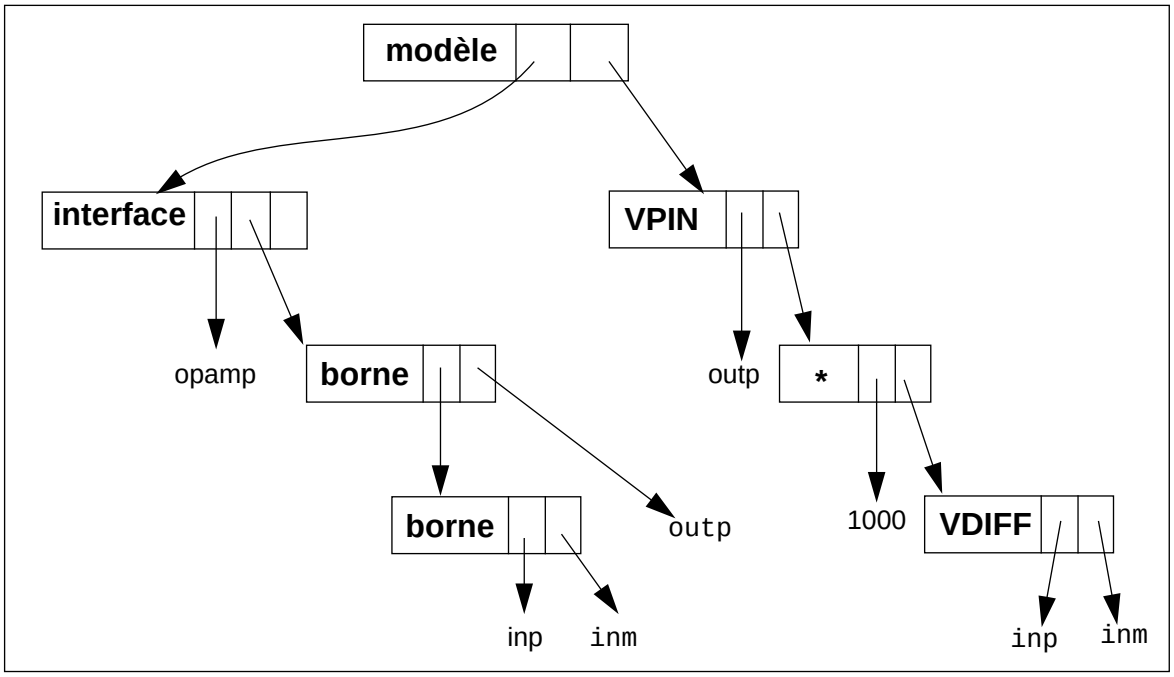


Figure 59 Arbre abstrait partiel associé à l'arbre syntaxique de ELDO-FAS

plusieurs champs pointant sur des noeuds fils. Ainsi, dans la Figure 59, le noeud nommé **VPIN** correspond à la seconde forme de la règle instruction **instr**. Il en est de même pour les différentes sortes d'expression, étiquetées ***** ou **VDIFF**. De plus, certaines règles simples, comme ***nom***, peuvent même être éliminées.

D'autre part, des *attributs* doivent être généralement associés aux noeuds comme le type des identificateurs et des expressions, pour un contrôle de type ultérieur. Des champs supplémentaires sont donc souvent nécessaires.

Pour générer une telle structure, des *actions* sont associées aux règles grammaticales et sont exécutées lorsque la règle est reconnue. Dans le cas de la règle **expr** précédemment décrite, il est nécessaire de définir un certain nombre de fonctions qui rendent un pointeur vers le noeud nouvellement créé. Le Tableau 16 suggère la forme de ces actions pour lesquelles:

- NB, ID, VDIFF, '*' désigne les étiquettes des noeuds ou feuilles créés
- `nombre.valeur` est la valeur numérique du nombre lu,
- `nom.entrée` est un pointeur sur l'identificateur nom qui est stocké dans une table des symboles (une telle table stocke tous les identificateurs trouvés et assure leur gestion),
- `expr.pnoeud` correspond à un pointeur sur noeud.

À la fin de l'analyse syntaxique du langage source, cette structure de données doit être traitée pour générer le modèle dans un autre format. Les opérations effectuées consistent par

Tableau 16 Fonctions associées aux expressions

Règle syntaxique	Action associée
`expr::= nombre`	`créerFeuille(NB,nombre.valeur)`
`expr::= nom`	`créerFeuille(ID,nom.entrée)`
`expr::= VOLT.DIFF(`nom_1 `,`nom_2`)`	`créerNoeud(VDIFF,`nom_1`.entrée,`nom_2`.entrée)`
`expr::=` $expr_1$ `*` $expr_2$	`créerNoeud('*',`$expr_1$`.pnoeud,`$expr_2$`.pnoeud)`

exemple en une réorganisation des informations, en la création d'instructions et de variables supplémentaires et enfin en un contrôle des identificateurs pour éviter tout conflit avec les mots clés du langage cible.

Notons que pour faciliter l'écriture l'analyseurs, des outils très efficaces ont été développés et sont disponibles dans l'environnement *Unix*. Ainsi le programme *Lex* permet de générer automatiquement un analyseur lexical à partir de la description des unités lexicales du flot d'entrée. Dans le domaine des analyseurs syntaxiques, l'outil *Yacc* (*Yet Another Compiler Compiler*) construit automatiquement un analyseur syntaxique de grammaire LALR, correspondant à la plupart des langages de programmation, à partir de la spécification des règles grammaticales du langage. La syntaxe utilisée pour cette description est très proche de la forme BNF précédemment utilisée.

Ce dernier outil a d'ailleurs été utilisé avec succès pour le développement du premier traducteur FAS->MAST pour lequel la structure intermédiaire de données est basée sur une suite de listes chainées. D'autre part, *Yacc* est à la base de l'outil de génération automatique de traducteurs, développé à SGS-Thomson et nommé *Zébu* [SaF93], et qui a été utilisé pour le développement des traducteurs FAS->CFAS puis FAS->HDL-A.

2.2 Environnement de conception de traducteurs: Zébu

Cet outil a été initialement développé dans le but de faciliter la réalisation de traducteurs dans le domaine des langages de description de layout, par exemple entre les diverses versions de *EDIF* et les sous-ensembles de description structurelle de *VHDL* et *Verilog* [SaF93]. Pour cela, le code permettant de construire la structure intermédiaire de données est automatiquement généré en utilisant une bibliothèque de fonctions orientées objet (C++). Cependant, dans le domaine des langages comportementaux analogiques, qui possédent des

grammaires relativement complexes, de nombreux traitements de données restent à effectuer.

2.2.1 Principe

Soit par exemple deux langages A et B. *Zébu* crée automatiquement les traducteurs *A->B et B->A* ainsi que des programmes de remise en page *A->A* et *B->B*, simplement à partir de trois fichiers définis par l'utilisateur et décrivant: (a) la *structure de l'arbre abstrait* intermédiaire ou *spécification*, (b) la *grammaire* de A et (c) celle de B (cf Figure 60). Notons de plus que la description de la grammaire est associée à celle du *format* de mise en page.

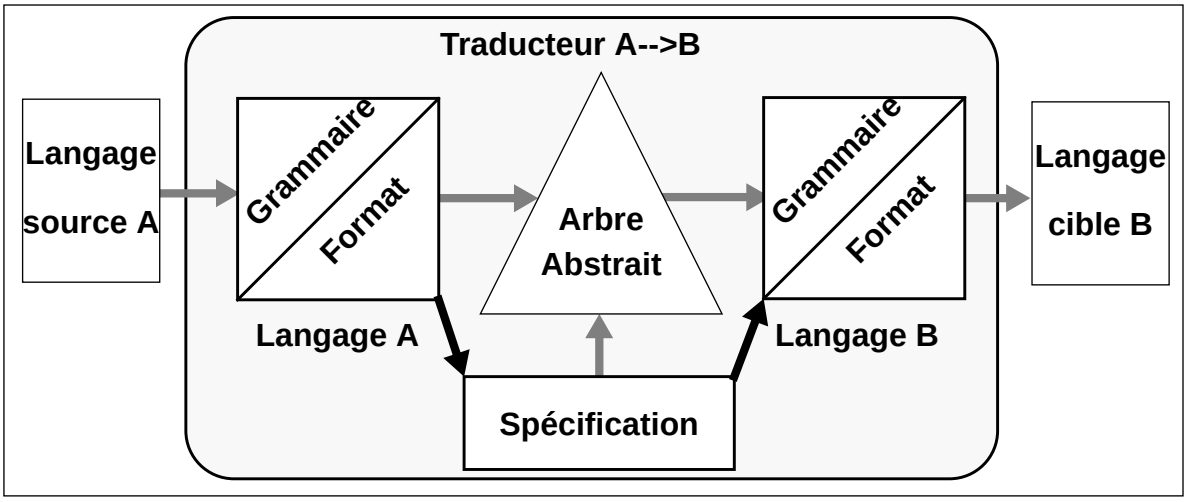


Figure 60 Principe du traducteur A-->B avec *Zébu*

a) Spécification de l'arbre abstrait

Cette description définit les noms ou étiquettes des divers noeuds de l'arbre ainsi que sa structure hiérarchique. C'est elle qui permet d'établir la correspondance entre les deux langages. Si l'on reprend l'exemple du paragraphe précédent, cette spécification pourrait avoir la forme suivante:

```
modele(interface,init,code)
interface(nom,borne,declare)
declare(borne)
borne(borne,nom)
code(bloc)
bloc(bloc,instr)
EGAL(nom,expr)
VPIN(nom)
VDIFF(NOM1,NOM2)
MULT(EXPR1,EXPR2)
ID(nom)
NB(nombre)
```

```
...
```

Ainsi, la ligne `modele(interface,init,code)` définit le noeud `modele` (racine de l'arbre) ayant pour fils les variables `interface, init et code`.

Par défaut, l'étiquette est le nom de la règle, mais en cas de choix multiple, par exemple pour les règles **instr** et **expr**, des noms différents doivent être donnés pour distinguer les diverses productions. Ainsi, **instr** pourra avoir les étiquettes **EGAL** ou **VPIN**, et **expr** sera associé à **VDIFF** et **MULT**.

Cependant, cette distinction n'est pas effectuée dans le cas des règles récursives comme **borne** et **bloc**, qui ont pourtant deux alternatives. En fait, le noeud qui correspond au premier élément trouvé et qui serait défini par **borne**::=***nom*** ou **bloc**::=**instr**, est inutile car il ne contient qu'un seul champ. L'information utile est plutôt ***nom*** ou l'étiquette de **instr**. Pour éviter la création d'un noeud intermédiaire, nous verrrons qu'un opérateur doit être utilisé lors de la description des règles. Il est nommé *opérateur de condensation* de l'arbre syntaxique.

De plus, lorsque deux variables d'une règle ont le même nom, deux étiquettes différentes doivent être spécifiées pour éviter toute collision (**NOM1** et **NOM2** dans la règle **VDIFF**).

Enfin, les règles simples, telles que **nom** et **nombre**, n'ont pas besoin de spécification. Notons que **nom** est un identificateur géré par une table des symboles.

Cette spécification permet à Zébu de générer automatiquement une bibliothèque de fonctions C++ de génération de noeud. À chaque noeud défini dans le fichier correspond une fonction appropriée admettant comme paramètres les divers noeuds fils.

b) Grammaire FAS

Pour réaliser la traduction, les descriptions des grammaires des deux langages doivent prendre en compte cette spécification commune. Ainsi, la grammaire de FAS devra être écrite de la façon suivante, avec une syntaxe simplifiée de *Zébu*:

```
modele     :    "AMODEL" interface init code "ENDMODEL" ;
interface  :    nom "(" borne ")" ";" declare ;
declare    :    "DECLARE" "PIN" borne ":" "ELECTRICAL" ";" ;
!borne     :    nom ;
borne      |    borne "," nom ;
code       :    "ANALOG" bloc "ENDANALOG" ;
!bloc      :    instr ;
bloc       :    bloc instr ;
EGAL[instr]     :    "MAKE" nom "=" expr ;
VPIN[instr]     |    "MAKE" "VOLT" "." "ACROSS" "(" nom ")" "=" expr ;
NB[expr]        :    nombre ;
ID[expr]        |    nom ;
```

```
VDIFF[expr]     |    "VOLT" "." "DIFF" "(" NOM1[nom] "," NOM2[nom] ")" ;
MULT[expr]      |    EXPR1[expr] "*" EXPR2[expr] ;
...
```

On observera les points suivants:

- les noms des étiquettes, lorsqu'ils sont spécifiés, précèdent les noms des règles ou des variables qui sont placés entre crochets [ ],
- le point d'exclamation **!** est un *opérateur de condensation* de l'arbre syntaxique pour la formation d'un arbre abstrait plus compact: il permet de supprimer le noeud correspondant à la production devant laquelle il est placé,
- tous les éléments terminaux sont placés entre doubles-quotes et un point virgule est placé en fin de chaque règle,
- aucune action n'est spécifiée (dans ce cas simple): les actions sont en effet automatiquement insérées par Zébu en fin de chaque règle et entre accolades { } (syntaxe *Yacc*), et appellent des fonctions C++ qui ont été construites à partir du fichier de spécification. Cependant, l'utilisateur a la possibilité de définir des actions spécifiques, concernant d'autres types de traitement de données. Cela a été indispensable dans le cadre de cette traduction de langages comportementaux.

c) Grammaire HDL-A

La grammaire HDL-A équivalente serait:

```
modele     :    interface code ;
interface  :    "ENTITY" <" "> nom <" "> "IS" <"%>\n">
                declare <"%<\n">
                "END" <" "> "ENTITY" ";" <"\n">
                "ARCHITECTURE" <" "> ARCH[nom] <" "> "OF"
                <" "> nom <" "> "IS" <"\n"> ;

declare    :    "PIN" "(" borne ":" <" "> "ELECTRICAL" ")" ";" ;
!borne     :    nom ;
borne      |    borne "," nom ;
code       :    "BEGIN" <" "> "RELATION" <"\n">
                "PROCEDURAL" <" "> "FOR" <" "> "DC" "," "AC" ","
                "TRANSIENT" <" "> "=>" <"%>\n">
                     bloc <"%<\n">
                "END" <" "> "RELATION" ";" <"\n">
                "END" <" "> "ARCHITECTURE" ";" ;
!bloc      :    instr ;
bloc       :    bloc instr ;
EGAL[instr]    :    nom ":=" expr ";" <"\n"> ;
```

```
VPIN[instr]     |    nom "." "V" "%=" expr ";" <"\n"> ;
NB[expr]        :    nombre ;
ID[expr]        |    nom ;
VDIFF[expr]     |    "[" NOM1[nom] "," NOM2[nom] "]" "." "V" ;
MULT[expr]      |    EXPR1[expr] "*" EXPR2[expr] ;
...
```

Notons que:

- les chaînes de caractères qui sont placées entre crochets < > ne sont utilisées que pour la mise en page du langage cible sur le flot de sortie. C'est pourquoi nous les avons ignorées pour la grammaire FAS: <" "> introduit un espace, <"\n"> effectue un retour à la ligne et <"%>\n"> crée en plus une tabulation (<"%<\n"> la supprime).
- la règle `interface` nécessite en HDL-A un champ suppémentaire `ARCH[nom]` correspondant au nom de l'architecture. La spécification `interface(nom,borne,declare)` doit donc être modifiée en `interface(nom,ARCH,borne,declare)`.La valeur de **ARCH** pourra être une constante, "`ideal`" par exemple, et sera définie par une action spécifique de la règle `interface` de la grammaire FAS.

Le Tableau 17 présente le résultat de la traduction ainsi spécifiée.

Tableau 17 Résultat de la traduction FAS-->HDL-A

Modèle FAS	Modèle HDL-A

`AMODEL opamp(inp,inm,outp);`
`DECLARE PIN inp,inm,outp: ELECTRICAL;`
	`ENTITY opamp IS`
	`  PIN(inp,inm,outp: ELECTRICAL);`
	`END ENTITY;`

`ANALOG`
`  make volt.across(outp)=`
`  1000*volt.diff(inp,inm)`
`ENDANALOG`
`ENDMODEL`
	`ARCHITECTURE ideal OF opamp IS`
	`BEGIN RELATION`
	`PROCEDURAL FOR DC,AC,TRANSIENT =>`
	`  outp.V %= 1000*[inp,inm].V;`
	`END RELATION ;`
	`END ARCHITECTURE ;`

2.2.2 Difficultés

Alors que l'un des buts de l'environnement *Zébu* de conception de traducteurs est d'éviter toute programmation, de nombreuses manipulations de données sont en fait à réaliser pour les langages comportementaux car la définition d'une structure commune est très difficile à établir.

Prenons tout simplement le cas de la traduction des déclarations de variables de FAS en

HDL-A. En HDL-A, connexions et paramètres sont déclarés dans la partie *entité* du modèle et dans un certain ordre (paramètres, connexions digitales, couplages, connexions analogiques), alors que les variables locales sont déclarées dans l'*architecture* (cf Tableau 18). De plus, les paramètres doivent être initialisés dans l'ordre de leur déclaration. La grammaire FAS précédemment décrite n'est donc pas adaptée à HDL-A et un tri des données est en fait nécessaire.

Tableau 18 Traduction réelle FAS-->HDL-A

Modèle FAS	Modèle HDL-A

`AMODEL opamp( inp, inm, out) ;`
`  DECLARE PIN inp,inm: ELECTRICAL;`
`  DECLARE LOCAL vin: REAL;`
`  DECLARE PARAM voff,gain:REAL;`
`  DECLARE PIN out: ELECTRICAL;`
	`ENTITY opamp IS`
	`  GENERIC( voff, gain: REAL);`
	`  PIN(inp, inm, out_: ELECTRICAL);`
	`END ENTITY opamp;`

	`ARCHITECTURE ideal OF opamp IS`
	`  STATE vin: ANALOG;`
	`  BEGIN RELATION`

`INITIALIZE`
`  make gain = 1e6`
`  make voff = 1e-6`
`  VSOURCE(outp)`
`ENDINITIALIZE`
	`  PROCEDURAL FOR INIT =>`
	`  voff := 1.0e-6;`
	`  gain := 1.0e6;`

`ANALOG`
`  make vin=volt.diff(inp,inm)+voff`
`  make volt.across(out)=gain*vin`
`ENDANALOG`
`ENDMODEL`
	`  PROCEDURAL FOR DC,AC,TRANSIENT =>`
	`  vin := [inp,inm].V+voff;`
	`  out_.V %= gain*vin;`
	`  END RELATION ;`
	`END ARCHITECTURE ideal;`

Ce type de problème a cependant été résolu à l'aide de la bibliothèque très complète de fonctions C++ que *Zébu* construit automatiquement à partir du fichier de spécification de l'arbre abstrait. Par exemple, des fonctions de lecture et d'écriture des valeurs de noeuds, d'ajout et de lecture d'attributs sont aujourd'hui disponibles. Les paragraphes suivants présentent d'ailleurs plusieurs exemples permettant d'appréhender les techniques utilisées au sujet de la déclaration des variables, de la définition des modes d'analyse et de l'utilisation des équations implicites.

3 Déclaration des variables (FAS/C-FAS/HDL-A)

3.1 Traduction FAS->C-FAS

Le langage C-FAS étant l'interface C de FAS, il est très proche de FAS en ce qui concerne la méthodologie de description analogique. Les propriétés de ce traducteur sont présentées en détail en Annexe 8 et nous nous limiterons ici au problème de la traduction des variables. En effet, les deux langages n'adoptent pas la même approche quant à la gestion des différents types de variables: celle de FAS est de haut-niveau car des types élaborés y sont définis alors que celle de C-FAS se base sur une représentation de bas-niveau par l'utilisation de vecteurs.

Les différents types de variables du langage FAS sont les suivants:

- variables ***locales (local)***: ce sont des variables intermédiaires dont la valeur n'est pas conservée d'une itération à l'autre du simulateur.
- variables ***analogiques (state, istate)***: leurs valeurs sont mémorisées au cours de la simulation, ce qui permet d'accéder à leurs valeurs passées, de leur appliquer les opérateurs de dérivation et d'intégration et de les afficher en fin de simulation. Les variables ***istate*** sont, par rapport aux ***state***, des inconnues résolues à l'aide d'équations implicites ***solve***.
- ***paramètres (param)***: ils permettent de décrire des modèles génériques. Les valeurs par défaut sont précisées dans la partie initialisation.
- ***constantes (usep)***: leurs valeurs sont déterminées à partir des paramètres dans la partie d'*initialisation* uniquement.
- deux types de ***connexions*** analogiques: les bornes (***pin***) qui peuvent appartenir à différents domaines physiques (***electrical***, ***mechanical***,...) et les couplages (***ic***).

Le langage C-FAS définit quant à lui des vecteurs nommés ***State[], Param[], Usep[]*** qui contiennent les valeurs respectivement des variables analogiques, des paramètres et des constantes. Ces variables sont d'ailleurs désignées simplement par leur index dans des fonctions C-FAS. De plus, alors que les variables FAS peuvent être de sous-type *real*, *integer* ou *boolean*, les vecteurs correspondants C-FAS sont déclarés comme étant de type réel (*double*).

De la même manière, les points de connexion, ***pin*** et ***ic***, sont repérés par un index qui correspond à leur rang respectif dans la ligne d'appel du modèle.

Seules les variables locales sont en fait utilisées de la même façon que dans tout langage de programmation.

Les objectifs du traducteur FAS-->C-FAS concernant la traduction des variables FAS sont donc principalement:

- de générer l'***indexation*** des variables FAS *param, usep, state, istate, pin, ic* lors de leur déclaration: pour augmenter la lisibilité du modèle C-FAS final, les commandes de définition de macro-fonctions du pré-processeur C sont utilisées: `#DEFINE NOM INDEX` permet d'associer le nom `NOM` à l'index (entier) `INDEX`. Ne sont ensuite manipulés que les noms des variables en majuscules.
- d'introduire des fonctions C de ***conversion de type*** dans les expressions pour éviter tout problème de typage, en particulier pour les expressions booléennes et l'indexation des vecteurs par des entiers.
- de définir les ***valeurs par défaut des paramètres***: des instructions particulières sont utilisées par le traducteur pour tenir compte des deux modes d'appel des modèles C-FAS, soit en définissant une liste de paramètres pour chaque occurence du modèle, soit en définissant une liste de paramètres qui est commune à plusieurs occurences. L'Annexe 1 donne plus de détails à ce sujet.

Pour effectuer ces opérations, le traducteur doit stocker, pour chaque variable définie dans le modèle source, le type (*State, Param, Usep...*) et le sous-type (*real, integer, boolean*). Ceci est réalisé par *Zébu* par la création d'une ***liste d'attributs*** pour chaque élément terminal, ou identificateur, de l'arbre abstrait. Pour illustrer les possibilités du traducteur, voici le modèle C-FAS correspondant au code FAS du Tableau 18:

```
#include ...
#define INP 1
#define INM 2
#define OUT 3
#define VOFF 1
#define GAIN 2
opamp FASARGU {
  double vin;
  if ((MODE == ALLOCATE) || (MODE == INIT_TEMP)) {
       double p[2];
       if (Model!=NULL) {
            define_equivalence_model("GAIN","P2");
            define_equivalence_model("VOFF","P1");
            if (MODE==ALLOCATE) {
                 Init_Model_Param("P2",1e6);
                 Init_Model_Param("P1",1e-6);
```

```
                }
                p[1]=Model->p2;
                p[0]=Model->p1;
          } else {
                p[1]=1e6;
                p[0]=1e-6;
          }
          order_param("GAIN",GAIN);
          order_param("VOFF",VOFF);
          VStatus(OUTP, GND);
          if (MODE == INIT_TEMP){
                default_param("GAIN",p[1]);
                default_param("VOFF",p[0]);
          }
     }
     if ((MODE == TRANSIENT) || (MODE == DC)) {
          vin=VDiff(INP,INM)+(double)Param[VOFF];
          VAcross(OUT, GND, (double)Param[GAIN]*vin);
     }
}
```

où:

- le bloc défini par **`if ((MODE == ALLOCATE) || (MODE == INIT_TEMP)){...}`** correspond à l'initialisation des paramètres et des constantes: la signification des diverses fonctions qui y sont utilisées est donnée en Annexe 8.
- le bloc défini par **`if ((MODE == TRANSIENT) || (MODE == DC)){...}`** correspond à la description comportementale du modèle,

3.2 Traduction FAS->HDL-A

Les propriétés du traducteur FAS->HDL-A sont présentées en détail en Annexe 7. Nous abordons ici le problème de traduction de la déclaration des variables, qui a été déjà introduit au paragraphe 2.2.2. La méthode suivie a consisté à adapter les règles de grammaire en fonction des possibilités de *Zébu* quant à la manipulation des données et à définir un langage intermédiaire, nommé REF, pour diviser la traduction en deux étapes:

- ***FAS-->REF*** concerne principalement la déclaration des variables FAS qui doivent être triées en fonction de leur type,
- ***REF-->HDLA*** traduit surtout le code du modèle et crée les différents blocs associés aux trois modes d'analyse. Cette traduction fait l'objet du paragraphe suivant.

Pour décrire l'action du module FAS-->REF, nous prendrons encore l'exemple du Tableau 18. Le modèle intermédiaire en REF est le suivant:

```
MODEL opamp {
       PARAMETER {
            voff,gain : REAL
       } PIN {
            inp,inm : ELECTRICAL
            out : ELECTRICAL
       } LOCAL {
            vin : REAL
       } DESCRIPTION {
            INITIALIZATION {
                 voff = 1e-6
                 gain = 1e6
                 VSOURCE(outp)
            } ANALOG {
                 vin = V(inp,inm)+voff
                 V.ACROSS(PIN=out EXPR=gain*vin)
            }
       }
}
```

La description des équations analogiques en REF est assez proche de la decription FAS. Cependant, la déclaration des variables est plus structurée. En FAS, il est en effet possible d'utiliser plusieurs instructions de déclaration pour un même type de variables: c'est par exemple, le cas des bornes (*pin*) du Tableau 18. Au contraire, REF définit un seul bloc de déclaration par type: `PARAMETER{}`, `PIN{}`, `LOCAL{}` mais aussi `COUPLING{}`, `CONSTANT{}`, `STATE{}` et `UNKNOWN{}`. Ainsi, il est plus facile pour REF-->HDL-A de scinder les déclarations en deux groupes: celles qui concernent l'entité du modèle (paramètres, couplages et bornes, qui doivent être déclarés dans cet ordre) de celles qui sont propres à l'architecture (constantes, locales, variables analogiques).

Pour obtenir ce résultat, une grammaire particulière doit être décrite pour FAS, REF et HDL-A, afin d'autoriser les deux modes de déclarations FAS et HDL-A. La grammaire ***FAS*** est la suivante:

```
DeclareList: DeclareList Declare ;
!Declarelist: Declarelist[Declare1];
Declare1: Declare;

Declare: "declare" Type Objet ";" { fas2refDec(Type, Objet); } ;
Paramdec[Type]:     "param" ;
Coupledec[Type]:    "ic" ;
Pindec[Type]:       "pin" ;
Localdec[Type]:     "local" ;
Cstedec[Type]:      "usep" ;
Statedec[Type]:     "state" ;
```

```
Istatedec[Type]:    "istate" ;

Simple_list[Objet]: Grouplist;
Grouplist: Element ;
Element: Idlist ":" Subtype ;

Idlist: Idlist "," ident ;
!Idlist: Idlist[ident1] ;
ident1: ident ;
Var[ident]: name ;
Array[ident]: name "(" First[int] ":" Last[int] ")" ;

Boolean[Subtype]:   "boolean" ;
Integer[Subtype]:   "integer" ;
Real[Subtype]:      "real" ;
Elec[Subtype]:      "electrical" ;
...
```

où:

- `DeclareList` est la liste des déclarations notées `Declare`
- `Declare` est défini par le type (`Dectype`) et un champ (`Objet`) qui fait référence à la liste `Grouplist` composées d'un élément déclaratif simple (`Element`)
- `Element` se compose d'une liste de variables (`Idlist`) et d'un sous-type (`Subtype`)

La structure de la grammaire de ***REF*** est légèrement différente. Les différences concernent surtout:

- l'instruction de déclaration (`Declare`) définit un bloc et devient:

  ```
  Declare: Type "{" Objet "}" ;
  ```

- la liste `Grouplist` de REF peut être composée de plusieurs instructions de déclaration (`Element`):

  ```
  Grouplist: Grouplist Element ;
  !Grouplist: Grouplist[Element1] ;
  Element1: Element;
  ```

  où `Element1` désigne le premier élément.
- de nouveaux sous-types sont introduits pour la déclaration des vecteurs:

  ```
  VReal[Subtype]: "REAL_VECTOR" "(" First[int] ":" Last[int] ")" ;
  VElec[Subtype]: "ELECTRICAL_VECTOR" "(" First[int] ":" Last[int] ")" ;
  ...
  ```

Quant à la grammaire ***HDL-A***, elle doit différencier la déclaration des objets de l'interface (*generic*, *coupling*, *pin*) des objets internes (variables analogiques, locales,...). Ceci

est simplement réalisé en décrivant les deux types de structure:

```
Simple_list[Objet]: Grouplist ;
Parenthese_list[Objet]: "(" Grouplist ")" ;
```

où **`Parenthese_list`** concerne uniquement les déclarations de l'interface. Bien sûr, les mots clés des autres règles doivent être aussi modifiées.

Enfin, la fonction **`fas2refDec`**`(Type,Objet)` associée à la règle `Declare` de la grammaire initiale FAS a pour but de regrouper les déclarations FAS en blocs en générant des listes intermédiaires pour chaque type trouvé. Ces diverses listes sont ensuite "raccrochées" à l'arbre du modèle source au niveau de la définition de l'interface.
Elle effectue aussi un traitement spécifique des vecteurs. Par exemple, la déclaration FAS suivante:

```
declare param voff,pole(1:10),gain:real;
```

devient l'instruction REF:

```
PARAMETER {
        voff,gain : REAL
        pole : REAL_VECTOR(1:10)
}
```

Et la déclaration correspondante HDL-A est:

```
GENERIC (voff,gain : REAL; pole : REAL_VECTOR(1 TO 10));
```

Un autre rôle de cette traduction FAS-->REF est le ***tri des définitions des valeurs par défaut des paramètres*** selon l'ordre de leur déclaration. Pour ce faire, une liste globale, générée lors de l'analyse syntaxique des déclarations, est utilisée comme référence après analyse du bloc d'initialisation: pour chaque variable de cette liste, le bloc d'initialisation est parcouru pour chercher une éventuelle définition. Si une telle instruction est identifiée, elle est recopiée dans un nouveau bloc qui sera finalement substitué au bloc originel d'initialisation.

4 Traduction FAS->HDL-A des modes d'analyse

Alors que nous avons déjà traité de la traduction des déclarations de variables au paragraphe 3.2, celui-ci porte principalement sur la traduction des différents domaines d'interprétation d'un modèle FAS en langage HDL-A.

Dans un modèle FAS, la description analogique est éventuellement divisée en plusieurs domaines d'interprétation par utilisation d'une instruction conditionnelle classique du type *IF-THEN-ELSE* et en testant la valeur d'une variable du simulateur, nommée *MODE,* qui vaut soit *DC* (analyse du point de fonctionnement), soit *TRANS* (analyse temporelle dite transitoire), soit *AC* (analyse fréquentielle).

L'*architecture* HDL-A peut comporter, quant à elle, une partie digitale, notée *PROCESS*, et une partie analogique, nommée *RELATION*. Cette dernière est elle-même subdivisée en plusieurs blocs distincts qui sont de deux types:

- les procédures (***procedural***) qui regroupent les équations explicites et sont classées en fonction de leur domaine d'interprétation: initialisation (*init*), étude DC (*dc*), AC (*ac*) et temporelle (*transient*),
- les blocs d'équations implicites (***equation***) qui sont eux-aussi associés aux différents modes d'analyse. La liste des inconnues à résoudre par le simulateur doit être de plus définie. Notons que le problème de traduction des équations implicites sera abordé au paragraphe suivant.

En ce qui concerne la définition des modes d'analyse, le rôle du traducteur est donc d'identifier le mode d'analyse courant de chaque instruction FAS pour les regrouper dans les différentes procédures HDL-A (ou blocs d'équations). Cette fonction est basée sur une pile qui stocke les définitions successives de mode au cours de l'analyse syntaxique. Pour insérer les nouveaux états, la règle conditionnelle *if-then-else* est définie en FAS (en réalité en REF) de façon à introduire deux étapes intermédiaires: la première concerne l'analyse de l'expression conditionnelle et la seconde la reconnaissance du mot clé *else*. La grammaire doit donc être décrite sous la forme suivante (avec la syntaxe de *Zébu*):

```
code  :    "ANALOG" bloc "ENDANALOG" { analyse_analogique(bloc); } ;
bloc  :    bloc instr1 ;
!bloc :    bloc[instr1] ;
instr1:    instr { analyse_instruction(instr); } ;
IF[instr]:          IF_KEY "THEN"
                         THEN_BLOC[bloc]
                    "ENDIF" ;
IF_ELSE[instr]:     IF_KEY "THEN"
                         THEN_BLOC[bloc]
                    ELSE_KEY
```

```
                        ELSE_BLOC[bloc]
                    "ENDIF" ;
IF_KEY:         "IF" expr     { analyse_mode_if(); };
ELSE_KEY:       "ELSE"        { analyse_mode_else(); };
MODE[expr]:     "MODE" ;
DC[expr]:       "DC"          { dc_mode=1; // variable globale } ;
TR[expr]:       "TRANS"       { tr_mode=1; // variable globale } ;
AC[expr]:       "AC"          { ac_mode=1; // variable globale } ;
TEST[expr]:     Expr1[expr] "=" Expr2[expr] ;
...
```

Les procédures sont définies dans la grammaire HDL-A de la manière suivante:

```
PROC[instr]: "PROCEDURAL" <" "> "FOR" <" "> Domainlist <" ">
                "=>" <"\n"> bloc ;
Domainlist:     Domainlist "," domain ;
!Domainlist:    Domainlist[domain1] ;
domain1:        domain;
DC[domain]:     "DC" ;
TR[domain]:     "TRANSIENT" ;
AC[domain]:      "AC" ;
Init[domain]:    "INIT" ;
```

où:

- la règle correspondant à la liste des instructions, `bloc`, est décrite de façon à ce que tout élément `instr1` ait le même format `bloc` (que ce soit le premier ou les suivants),
- la valeur des variables globales `dc_mode`, `tr_mode` et `ac_mode` passe à 1 lorsque les mots clés `DC`, `TRANS` ou `AC` sont reconnus par l'analyseur,
- les fonctions `analyse_mode_if()` et `analyse_mode_else()` sont exécutées respectivement lorsque les règles `IF_KEY` et `ELSE_KEY` ont été parcourues. En particulier, `analyse_mode_if()` définit un nouvel état de la pile en stockant un triplet constitué de la valeur des variables globales `dc_mode`, `ac_mode` et `tr_mode`, *même si aucun mode n'a été défini*. Enfin, les variables sont remises à 0.

  La fonction `analyse_mode_else()` stocke aussi un nouvel état dans la pile. Dans le cas où il existe un état précédent non-nul dans la pile, la valeur stockée correspond à sa négation. Sinon, il s'agit d'une instruction *if-then-else* sans définition de mode et la valeur du nouvel état est un triplet nul.
- la fonction `analyse_instruction(instr)` est exécutée lorsqu'une instruction `instr` a été reconnue. Elle teste le format (étiquette) de l'instruction. Si cette étiquette a pour valeur `IF` ou `IF_ELSE`, alors une analyse particulière est effectuée pour créer les procédu-

res HDL-A en fonction de l'état de la pile.

- la fonction `analyse_analogique(bloc)` est exécutée à la fin de l'analyse syntaxique du bloc analogique qui est entièrement parcouru afin de vérifier que le bloc n'est constitué que de procédures. Si ce n'est pas le cas, cela signifie que la description analogique du modèle FAS a débuté sans définition de domaine d'interprétation. Le traducteur recopie par défaut toutes les instructions dans les procédures *dc, ac* et *transient*, en effectuant les regroupements nécessaires de procédures.
- la liste `Domainlist` de la grammaire HDL-A définit la liste des modes possibles (notons l'existence d'un mode d'initialisation `INIT`).

La Figure 61 illuste la manière de procéder sur un exemple simple. Les deux premières instructions d'affectation n'introduisent aucun nouvel état. Le premier état, de valeur 100, est stocké après le parcours de la branche `IF_KEY` lors duquel la définition du mode DC est reconnu. Lorsque la règle `ELSE` est analysée, l'état "inverse" 011 est rajouté. Enfin, quand la règle `IF_ELSE` est entièrement déduite, les deux états 011 et 100 sont supprimés et on retrouve la situation initiale.
De plus, au moment de la réduction de la règle `IF_ELSE`, les deux blocs `THEN_BLOC` et `ELSE_BLOC` sont décomposés en procédures: `THEN_BLOC` donne *procedural for dc => ...* et `ELSE_BLOC` est à l'origine de *procedural for ac => ...* et de *procedural for transient => ...*
Ces trois instructions sont ensuite regroupées en une seule liste de type `bloc` qui remplace l'instruction `IF_ELSE` courante (l'arbre de la Figure 61 correspondant à cette étape est schématique). Enfin, le traducteur analyse le bloc analogique dont les instructions sont recopiées dans les trois procédures *dc*, *ac* et *transient* (seule la procédure *dc* est représentée en Figure 61).

D'autre part, il est possible que des instructions `IF` ou `IF_ELSE` soient imbriquées. Comme la lecture du modèle source par *Yacc* est un processus récursif, les blocs `THEN_BLOC` ou `ELSE_BLOC` d'une instruction conditionnelle définissant un mode particulier peuvent donc avoir été eux-mêmes décomposés en une liste de procédures à une étape précédente. Il y a dans ce cas un regroupement des procédures existantes si bien qu'à la fin de l'analyse syntaxique, le modèle HDL-A ne comporte qu'un seul bloc pour chaque type d'analyse.

L'imbrication d'instructions conditionnelles définissant des modes dans des instructions

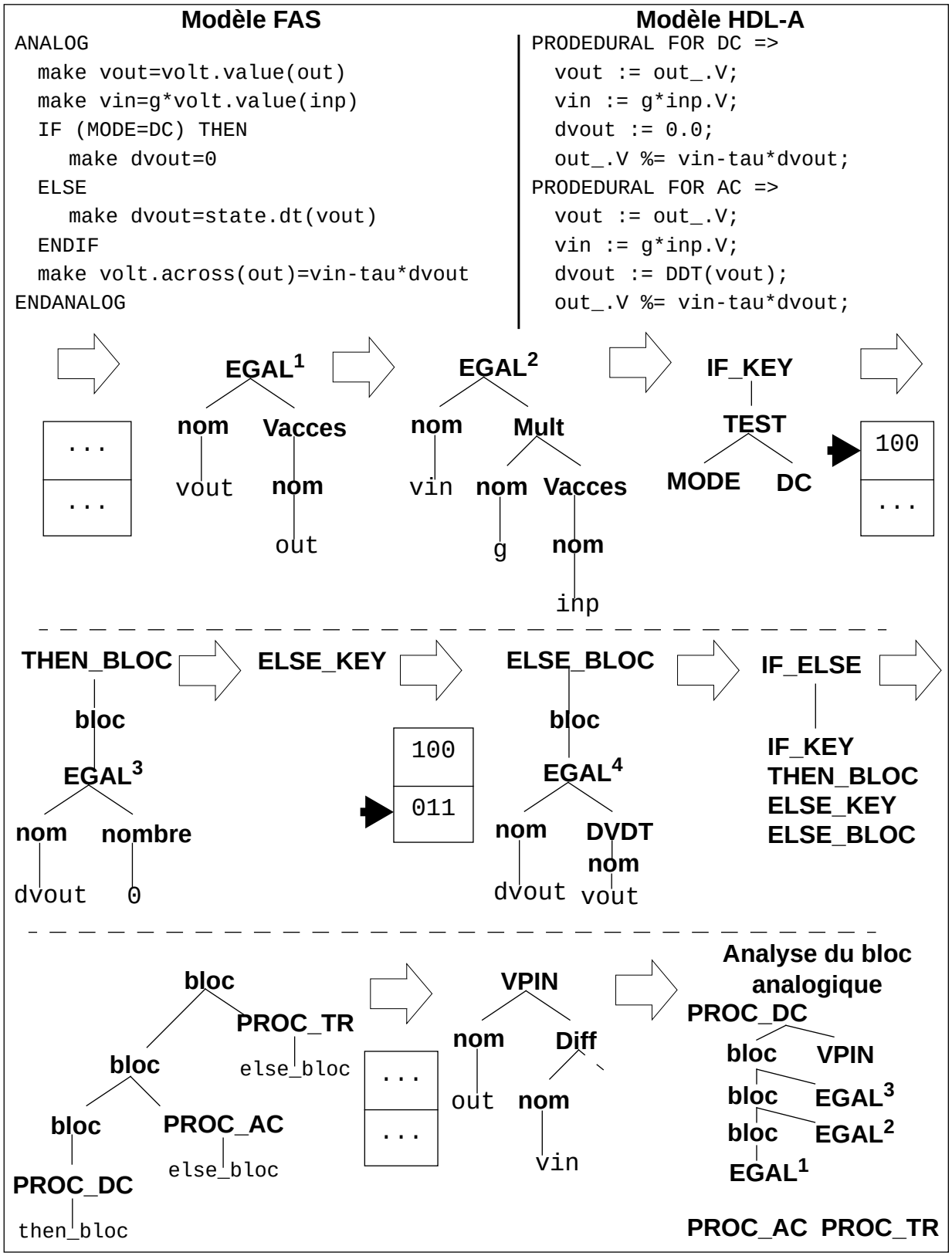


Figure 61 Analyse des modes d'analyse grâce à un système de pile

conditionnelles classiques est aussi prise en compte (cf Tableau 19). Au moment de la réduction de l'instruction conditionnelle simple externe, une instruction identique (`IF` ou

`IF_ELSE`) est définie dans chaque procédure existante. Le Tableau 19 montre l'étape intermédiaire correspondant au moment où la règle `IF_ELSE` a été reconnue.

Tableau 19 Imbrication d'instructions conditionnelles

FAS	Analyse de `IF level=1`	HDL-A

`IF level=1 THEN`
`  IF MODE=DC THEN bloc1`
`  ELSE bloc2`
`  ENDIF`
`ELSE`
`  IF MODE=DC THEN bloc3`
`  ELSE bloc4`
`  ENDIF`
`ENDIF`
	`IF level=1 THEN`
	`  PRODEDURAL FOR DC =>`
	`  bloc1`
	`  PRODEDURAL FOR AC =>`
	`  bloc2`
	`  (idem pour TRANSIENT)`
	`ELSE`
	`  PRODEDURAL FOR DC =>`
	`  bloc3`
	`  PRODEDURAL FOR AC =>`
	`  bloc4`
	`  (idem pour TRANSIENT)`
	`ENDIF`
		`PRODEDURAL FOR DC =>`
		`  IF level=1 THEN bloc1`
		`  ELSE bloc3`
		`  END IF;`
		`PRODEDURAL FOR AC =>`
		`  IF level=1 THEN bloc2`
		`  ELSE bloc4`
		`  END IF;`
		`PRODEDURAL FOR`
		`  TRANSIENT=>`
		`  IF level=1 THEN bloc2`
		`  ELSE bloc4`
		`  END IF;`

5 Équations implicites (FAS/HDL-A/MAST)

On distingue trois types d'équations pour exprimer les relations de branche d'un modèle analogique: les équations *explicites*, correspondant à celles utilisées dans tout langage de programmation classique, les équations *implicites*, qui nécessitent un certain nombre d'itérations pour leur résolution et enfin les équations de *connexion* sur les bornes obéissant aux lois de Kirchhoff (cf Chapitre 3).

En FAS, les *équations implicites* sont de la forme `solve(variable,expression)`, où `variable` est le nom de la variable implicite et `expression` est la partie gauche de l'équation implicite `expression=0`.

En HDL-A, elles sont regroupées dans des blocs particuliers, nommés `equation`, qui sont associés à une liste d'inconnues et à des modes d'analyse.

Enfin, dans la première version de MAST que nous avons étudiée [Mas90], des sections sont définies: les équations explicites sont regroupées dans une section nommée `values{}` alors que les équations implicites et de connexion sont utilisées dans une section nommée `equations{}`. Précisons que dans la version actuellement disponible [Mas93], ce n'est plus indispensable.

Ainsi, pour chaque instruction FAS définissant une équation implicite, les traducteurs FAS->HDLA et FAS->MAST effectuent les opérations suivantes:

- création d'une équation explicite qui affecte la valeur de l'expression de l'équation implicite à une nouvelle variable intermédiaire. Cette équation remplace l'équation implicite FAS.
- création et déclaration de la variable analogique définie précédemment,
- création d'une équation implicite dans le bloc d'équations implicites du langage cible. Notons qu'en HDL-A, le mode d'analyse est déterminé grâce au système de pile décrit au paragraphe 4.

Le Tableau 20 illustre ce processus de traduction dans le cas d'un simple modèle de résistance. Remarquons que la traduction des équations de connexion de FAS (`curr.on(ip)=...`) vers MAST (`i(ip)+=...`) fait de même intervenir des variables intermédiaires (`curron0` et `curron1`). On se reportera en Annexe 7, Annexe 8 et Annexe 9 pour des explications détaillées sur les traducteurs FAS, HDL-A et MAST.

Tableau 20 Modèle d'une résistance en FAS, HDL-A et MAST

FAS	HDL-A	MAST

AMODEL res(ip,im) ;
declare pin ip,im: electrical;
declare param r: real;
declare local ddp: real;
declare istate ires: real;
INITIALIZE
  make r = 1
ENDINITIALIZE
ANALOG
  ddp = volt.diff(ip,im)
  solve(ires,ddp-r*ires)
  curr.on(ip) = ires
  curr.on(im) = -ires
ENDANALOG
 	
ENDMODEL
	ENTITY res IS
	  generic(r: real);
	  pin(ip,im: electrical);
	END ENTITY;
	
	ARCHITECTURE nom of res IS
	  variable ddp: analog ;
	  state ires: analog;
	  state expr_ires: analog;
	
	BEGIN RELATION
	PROCEDURAL FOR INIT =>
	  r := 1.0;
	PROCEDURAL FOR DC,AC,TRANSIENT =>
	  ddp := [ip,im].v;
	  expr_ires:=ddp-r*ires;
	  ip.I %= ires;
	  im.I %= -ires;
	EQUATION(ires) FOR DC,AC,
	TRANSIENT=>
	  (expr_ires) == 0.0;
	END RELATION ;
	END ARCHITECTURE ;
		TEMPLATE res ip im = r
		electrical ip, im
		number r = 1
		{
		val nu ddp,
		  expr0,
		  curron0,
		  curron1
		var nu ires
		VALUES {
		  ddp = (v(ip) - v(im))
		  expr0 = ddp - r*ires
		  curron0 = ires
		  curron1 = - ires
		}
		EQUATIONS{
		  ires: expr0 = 0
		  i(ip) += curron0
		  i(im) += curron1
		}
		}

6 Conclusion

Ce chapitre a permis de définir des méthodologies de traduction de langages comportementaux analogiques à l'aide de Yacc et de Zébu [SaF93]. Les principaux problèmes rencontrés lors du développement des traducteurs FAS vers CFAS, FAS vers HDL-A et FAS vers MAST ont été de plus décrits. La Figure 62 résume de manière schématique les principales étapes de l'élaboration d'un traducteur A vers B de langages comportementaux analogiques, avec un outil tel que Zébu: la spécification d'une structure commune de données, la description des grammaires de A et B, du format d'impression de B et enfin le développement de fonctions (ou actions) élaborées à partir d'une bibliothèque C++ générée par Zébu.

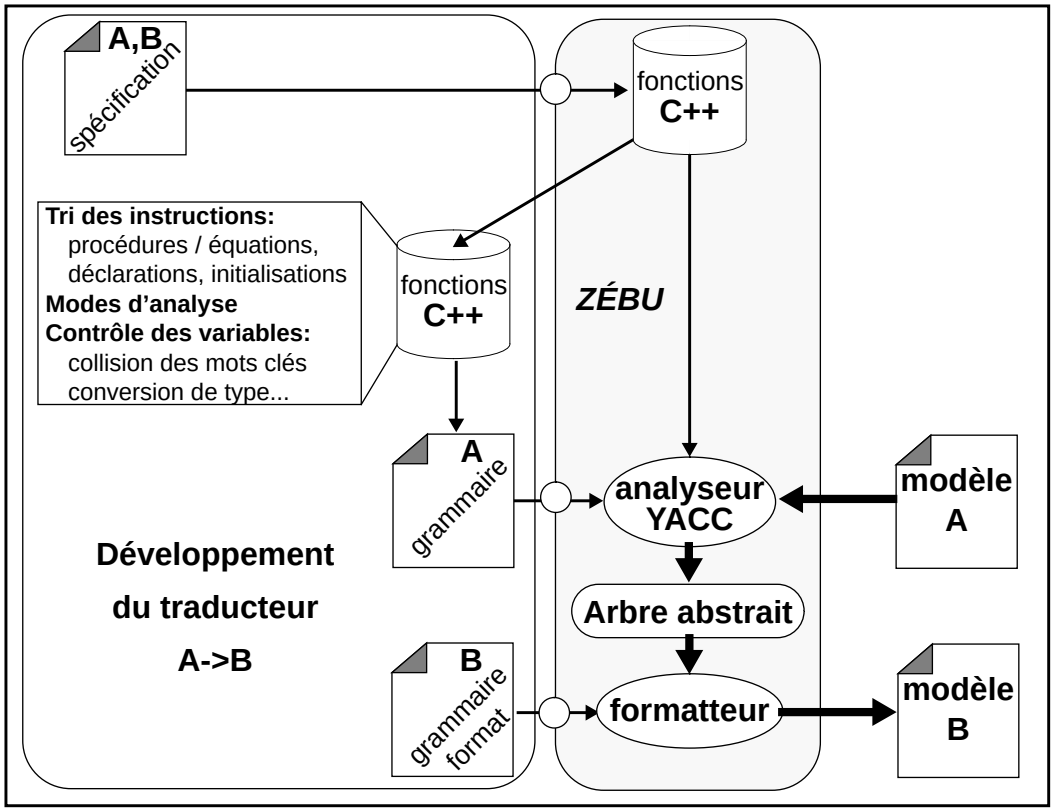


Figure 62 Résumé des actions de traduction avec Zébu

Même si certaines opérations de traduction restent à la charge de l'utilisateur, par exemple en MAST pour la description des fonctions non-linéaires, ou en HDL-A pour une analyse du type des expressions, les traducteurs qui ont été développés constituent une aide appréciable: FAS -> CFAS et FAS -> HDL-A ont permis de transférer avec succès une bibliothèque fonctionnelle de modèles FAS et FAS -> MAST a permis à SGS-Thomson de délivrer des modèles comportementaux MAST à un client du domaine automobile.

Chapitre 8: Conclusion

Cette thèse a permis de mettre en place un ensemble d'outils d'aide à la conception de modèles comportementaux de circuits analogiques et mixtes:

- une ***bibliothèque fonctionnelle de haut-niveau*** comportant des modèles analogiques/ digitaux adaptés aux études de systèmes. Elle propose en particulier de nombreuses solutions de modélisation comportementale avec le nouveau langage HDL-A. De plus, certains de ces modèles ont été utilisés avec succès pour la simulation d'un système électronique de déclenchement d'air-bag, à plusieurs niveaux hiérarchiques.
- une ***bibliothèque fonctionnelle de bas-niveau*** constituée de modèles analogiques dédiés à la création modulaire de macro-modèles d'amplificateurs opérationnels. Elle a été validée pour la modélisation d'un nouvel amplificateur de SGS-Thomson.
- un ***outil de caractérisation*** permettant de mesurer des performances complexes de circuits analogiques et de générer rapidement les paramètres du macro-modèle qui doit le représenter, à partir des données ainsi recueillies. Cette opération est particulièrement importante pour la validation d'un système après réalisation des blocs qui le constituent.
- une ***bibliothèque de procédures de mesure*** qui est actuellement limitée à l'étude des amplificateurs opérationnels. Ces procédures, qui sont principalement basées sur un schéma ou test-bench et des fonctions d'extraction, sont paramétrables, éventuellement hiérarchiques, et sont appelées par l'outil de caractérisation.
- une ***interface utilisateur*** intégrée à l'environnement de conception de SGS-Thomson et permettant au concepteur de définir ses propres procédures de mesure. Ces dernières sont ensuite sauvegardées dans une bibliothèque. Cette interface permet aussi de générer automatiquement des plans de caractérisation à partir d'un schéma-bloc de macro-modèle dont les paramètres sont définis en utilisant un nouveau langage de spécification de mesures, ou à partir d'un schéma dont les blocs font simplement appel aux procédures de mesure.
- des ***traducteurs automatiques de langages comportementaux***, dont le but est de faciliter

l'échange de modèles entre compagnies, en l'absence d'un standard de description analogique et mixte, ainsi que la migration de bibliothèques fonctionnelles vers de nouveaux langages. Sont disponibles les traducteurs FAS vers MAST, FAS vers CFAS et enfin FAS vers HDL-A.

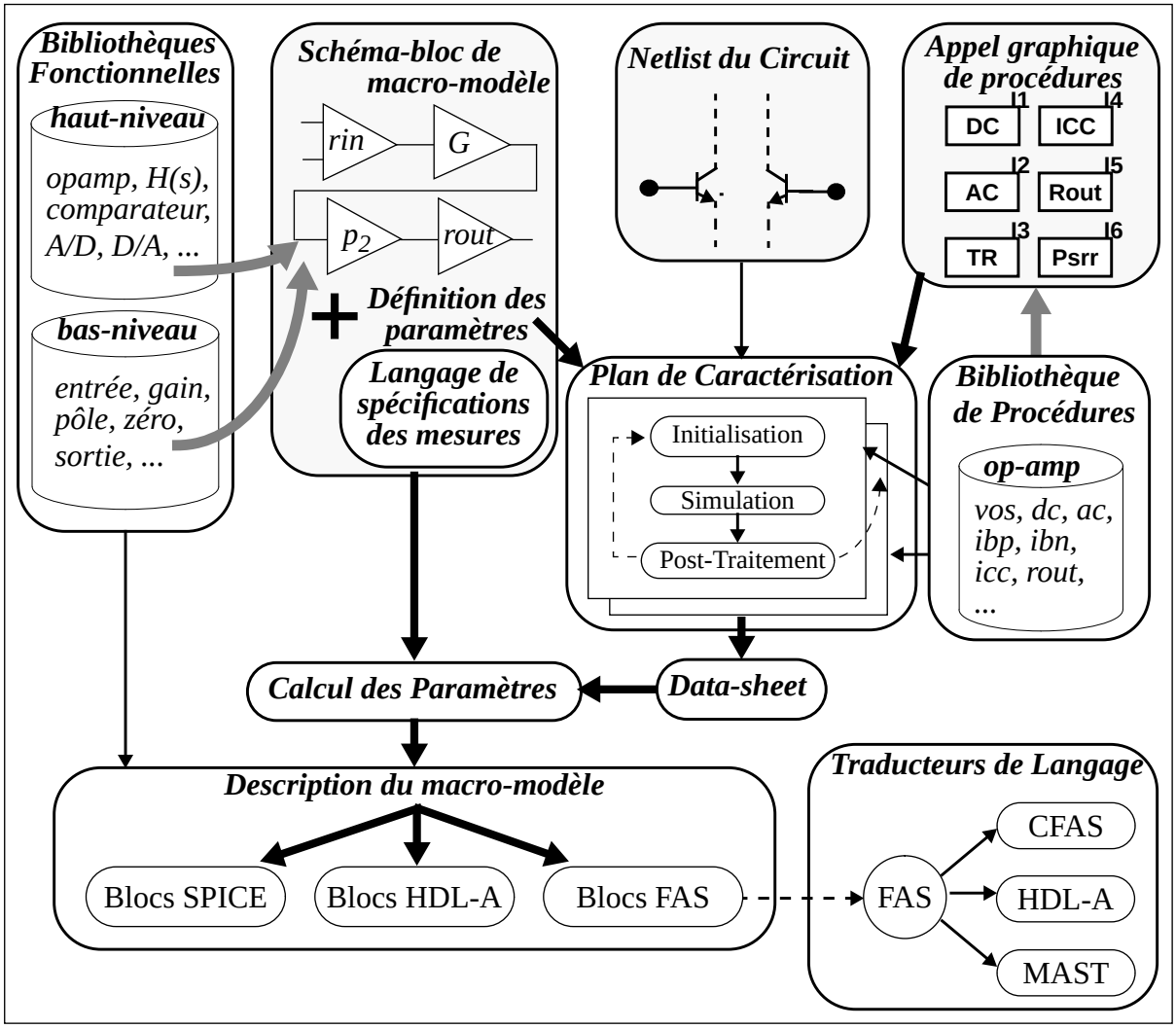


Figure 63 Environnement de conception de modèles comportementaux analogiques et mixtes

Cet environnement de conception de modèles comportementaux analogiques et mixtes est illustré en Figure 63. Il permet réellement d'étendre les fonctionnalités de l'environnement de conception analogique actuel de SGS-Thomson, en facilitant à la fois les méthodologies de conception descendante *top-down,* par l'offre de modèles comportementaux de haut-niveau, et les approches ascendantes *bottom-up* de validation, grâce à la bibliothèque de modèles de bas-niveau et à l'outil de caractérisation. L'expérience acquise dans le domaine de la traduction permet d'autre part d'assurer que les modèles comportementaux qui sont aujourd'hui développés dans un langage donné pourront être rapidement transférés dans un autre.

En fait, un tel environnement est particulièrement adapté au développement de plus en plus nécessaire de bibliothèques de cellules analogiques ou mixtes, complexes, telles que des convertisseurs ou des PLLs. En effet, même s'il n'a été validé que sur un circuit d'op-amp, cet environnement a été conçu pour être le plus général possible et il peut être appliqué à d'autres circuits et à des fonctions de plus haut-niveau, à deux conditions près: de nouvelles procédures de caractérisation devront certainement être développées, mais l'interface utilisateur est là pour faciliter cette tâche, ainsi que d'autres modèles, qui pourront cependant être élaborés en s'inspirant des éléments déjà présents dans les bibliothèques fonctionnelles.

D'autre part, l'outil de caractérisation peut être appliqué à n'importe quel niveau hiérarchique pour générer les paramètres de la représentation comportementale à partir de simulations de la description structurelle correspondante. Cette approche de caractérisation hiérarchique pourrait être automatisée et mériterait d'être étudiée sur une application concrète.

Cependant, pour connaître plus précisément les performances d'une architecture dans une phase de conception descendante ainsi que pour accorder une plus grande confiance au circuit final, l'élaboration de modèles comportementaux analogiques et mixtes, tenant compte de phénomènes réalistes du second ordre, tels que les non-linéarités, la distorsion, le bruit et les dérives statistiques des performances dues aux fluctuations de paramètres technologiques, devrait être maintenant étudiée. C'est en effet un sujet qui sera de plus en plus critique dans l'avenir du fait de l'accroissement de la densité d'intégration. Cela nécessite en particulier de développer de nouvelles techniques de modélisation comportementale, basées sur un langage standard tel que le futur VHDL-A, mais aussi de nouveaux outils de caractérisation et d'extraction de paramètres (pour des calculs statistiques et d'optimisation).

Avec de telles évolutions, l'environnement de modélisation comportementale, qui a été présenté dans ce document, constituerait un outil performant pour l'application de la méthodologie de conception descendante dirigée par les contraintes, définie à l'Université de Californie-Berkeley [Cha95] et qui vise "*l'augmentation de la probabilité d'obtenir le premier silicium correct et la réduction du temps de conception*". Les points complémentaires à étudier concernent l'étude de la testabilité à chaque étape de la conception, et le développement d'outils interactifs pour la synthèse de *layout*.

Références Bibliographiques

[AAS95] ***Analog Artist Schematics and Simulation***, ***Mixed-Signal Analysis***, Chapter 9, Training Manual, Version 4.3, Chapter 9, Cadence Design Systems Inc., January 11, 1995

[ABO90] R.Airiau, J.M.Bergé, V.Olive, J.Rouillard, ***VHDL: du langage à la modélisation***, Presses Polytechniques et Universitaires Romandes, 1990

[ACT92] ***Analog Characterization Tool***, Version 1.0.1, Reference Manual, SGS-Thomson Microelectronics, Central R&D, August 1992

[AlB90] M.Alexander, D.F. Bowers, Precision Monolithics Inc, ***New Spice compatible op-amp model boosts ac simulation accuracy***, Part 1, EDN February 15, 1990, pp. 143-164

[All86] P.E.Allen, ***A tutorial - Computer Aided Design of Analog Integrated Circuits***, Proc. IEEE Custom Integrated Circuits Conference, pp. 608-616, 1986

[AnE92] B.A.A.Antao, F.M.El-Turky, ***Automatic Analog Model Generation for Behavioral Simulation***, Proc. IEEE Custom Integrated Circuits Conference, pp. 12.2.1-12.2.4, 1992

[ARN70] R.Arnal, ***Signaux et Circuits***, Maîtrise d'électronique, d'électrotechnique et d'automatique, Signaux et Systèmes, Dunod Université, Paris, 1970

[Asp95] ***Aspire User's Manual***, ANACAD, Revison 1.0, February 1995

[ASU86] A.V.Aho, R.Sethi, J.D.Ullman, ***Compilers: Principles, Techniques and Tools***, Addison-Wesley 1986, ISBN 0-201-10088-6

[BCP74] G.R.Boyle, B.M.Cohn, D.O.Pederson, J.E.Solomon, ***Macromodeling of Integrated Circuit Operational Amplifiers***, IEEE Journal of Solid-State Circuits, Vol. 9, No. 6, December 1974

[BCU94] ***Behavioral Compiler User Guide***, Version 3.2a, Synopsis Inc, October 1994

[Bol94] P.Bolcato, ***Modélisation et simulation de bruit dans les circuits intégrés: analyse fréquentielle et régime transitoire***, Thèse de l'INPG, 26 Janvier 1994

[Cal91] D.J. Caldwell, ***Techniques let you write general-purpose Spice models***, EDN September 1991, p.149-154

[CaS91] G.Casinovi, A.Sangiovanni-Vincentelli, ***A Macromodeling Algorithm for Analog Circuits***, IEEE Transactions on Computer-Aided Design, Vol.10, No.2, February 1991

[Cfa93] ***CFAS: C-language FAS interface Manual***, Issue 1.0, ANACAD, August 1993

[Cha95] H.C-C.Chang, ***A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits***, Memorandum No UCB/ERL M95/21, University of California, Berkeley, April 1995

[ChL75] L.O.Chua, P-M.Lin, ***Computer Aided Analysis of Electronic Circuits: Algorithms & computational techniques***, Prentice-Hall Inc., Englewood Cliffs, New Jersey, pp. 111-120, 1975

[CoC92] J.A.Connelly, P.Choi, ***Macromodeling with SPICE***, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1992

[CLN94] H.Chang, E.Liu, R.Neff, E.Felt, E.Malavasi, E.Charbon, A.Sangiovanni-Vincentelli, P.R.Gray, ***Top-Down, Constraint-Driven Design Methodology Based Generation of n-bit Interpolative Current Source D/A Converters***, In Proc. IEEE Custom Integrated Circuits Conference, pp. 15.5.1-15.5.4, 1994

[DCB95] M.Degrauwe, M.Chevroulet, J.Bergqvist, JP.Bardyn, ***Microsystems: A Challenge for IC Designers***, In Proc. ESSCIRC'95, pp. 30-33, 1995

[DCT94] ***Design Compiler Tutorial***, Version 3.1a, Synopsis Inc, March 1994

[DFR94] ***Design Framework II Reference Manual***, 4.3, Cadence Design Systems Inc., March 1994

[DoD92] F.Dorel, M.Declerq, ***A Prototype for the Design Oriented Symbolic Analysis of Analog Circuits***, Proc. IEEE Custom Integrated Circuits Conference, pp. 12.5.1-12.5.4, 1992

[DOD94] IEEE VHDL subPAR 1076.1 ***Analog Extensions to VHDL Design Objective Document*** (DOD), Version 2.1, Nov. 8, 1994

[Eld94] ***ELDO User's Manual***, ANACAD, Issue 4.3, June 1994

[Fan83] San-Chin Fang, ***Analysis, computer simulation and properties of switched-capacitor networks***, Columbia Unervisity, 1983

[Fas92] ***ELDO-FAS Dynamical System Modeling User's Manual***, Version 4.1.x, ANACAD, July 1992

[Fer84] J.P. Ferrieux, ***Modélisation des convertisseurs continu-continu à découpage***, Thèse de l'INPG, 4 mai 1984

[FNS94] E.Felt, A.Narayan, A.Sangiovanni-Vincentelli, ***Measurement and Modeling of MOS Transistor Current Mismatch in Analog IC's***, in Proc. IEEE ICCAD, pp. 272-277, November 1994

[FRH91] F.V.Fernandez, A.Rodriguez-Vazquez, J.L.Huertas, ***Interactive AC Modeling and Characterization of Analog Circuits via Symbolic Analysis***, Analog Integrated Circuits and Signal Processing, Vol. 1,pp.183-208, Kluwer Academic Publishers, 1991

[FRM92] F.V.Fernandez, A.Rodriguez-Vazquez, J.J.Martin, J.L.Huertas, ***Accurate Simplification of Large Symbolic Formulae***, Proc. IEEE/ACM Int. Conf. on Comuter-Aided Design, pp. 318-321, November 1992

[Gaj89] D.Gajski, ***Silicon Compilation***, Addison Wesley, 1989

[GDW92] D.Gajski, N.Dutt, C.Wu, Y.Lin, ***High-Level Synthesis, Introduction to Chip and System Design***, Chapter 1, Kluwer Academic Publishers, 1992

[Gir91] M.Girard, ***Boucles à Verrouillage de Phase***, McGraw-Hill, Paris, ISBN: 2-7042-1157-4, 1991

[GSB93] C.Guardiani, P.Scandolara, J.Benkoski, ***Yield optimization of analog IC's using two-step analytic modeling methods***, IEEE Journal of Solid-State Circuits, Vol. 28, No. 7, July 1993

[GWS89] G.Gielen, H.Walscharts, W.Sansen, ***ISAAC: A Symbolic Simulator for Analog Integrated Circuits***, IEEE Journal of Solid-State Circuits, Vol. 24, No. 6, December 1989

[Hdl94] ***HDL-A Language Reference Manual***, Issue 1.0, ANACAD, July 1994

[Her93] D.B.Herbert, ***SPICE2 Voltage-Programmable Transfer Function Modeling***, R.Saleh and A.Yang , Editors, Circuits & Devices, pp.8-14, January 1993

[HeS85] B.Hennion, P.Senn, ***Eldo: A new third generation circuit simulator using the One-Step Relaxation method***, in Proceedings of ISCAS, pp. 1065-1068, 1985

[HGT91] S.A.Huss, M.Gerbershagen, G.Tränkle, ***Automatic Performance Characterization of Analog Functional Blocks***, Analog Integrated Circuits and Signal Processing, Vol. 1, pp. 277-286, Kluwer Academic Publishers, 1991

[HHN94] R.K.Henderson, M.Hinners, P.Nussbaum, L.Astier, ***Capture and Re-use of Analog Simulation Knowledge***, Proc. IEEE Custom Integrated Circuits Conference, pp. 15.2.1-15.2.4, 1994

[JEA89] Jeandel, ***Transfert de Masse et de Chaleur***, Cours de l'École Centrale de Lyon, p37, 1989

[JPO93] A.Jerraya, I.Park, K.O'Brien, ***AMICAL: An Interactive High-Level Synthesis Environment***, EDTC Paris-France, Feb. 1993

[JRS91] Y.C.Ju, V.B.Rao, R.A.Saleh, ***Consistency Checking and Optimization of Macromodels***, IEEE Transactions on Computer-Aided Design, Vol.10, No.8, August 1991

[Kar94] K.J. Karimi, ***Modeling and simulation of large DC power electronics systems***, Boing Computer Services, In Proceedings of the Conference on Modelling and Simulation 1994, p.1111-1115

[KeM94] U.Kemper, H.T.Mammen, ***Netlist and Behavioural Description of Macromodels for Analog Circuits***, Proc. Conference on Modelling and Simulation, June 1-3 1994, Barcelona, pp. 979-984

[KoC92] T.Koskinen, P.Y.K.Cheung, ***Hierarchical Tolerance Analysis using Behavioral Models***, Proc. IEEE Custom Integrated Circuits Conference, pp 3.4.1-3.4.4, 1992

[LCS93] E.Liu, H.C.Chang, A.Sangiovanni-Vincentelli, ***Analog System Verification in the Presence of Parasitics using Behavioral Simulation***, Proc. 30th ACM/ IEEE Design Automation Conference, pp. 159-163, 1993

[Liu91] R.W.Liu, ***A Circuit Theoretic Approach To Analog Fault Diagnosis,*** Testing and Diagnosis of Analog Circuits and Systems, edited by R.W. Liu, Van Nostrand Reinhold New-York, Chapter 1, 1991

[LiS92] E.Liu, A.Sangiovanni-Vincentelli, ***Behavioral Representation*****s *for VCO and Detectors in Phase-Lock Systems***, Proc. IEEE Custom Integrated Circuits Conference, pp. 12.3.1-12.3.4, 1992

[LKF94] E.Liu, W.Kao, E.Felt, A.Sangiovanni-Vincentelli, ***Analog Testability Analysis and Fault Diagnosis using Behavioral Modeling***, Proc. IEEE Custom Integrated Circuits Conference, pp. 17.3.1-17.3.4, 1994

[LSG91] E.Liu, A.Sangiovanni-Vincentelli, G.Gielen, P.R.Gray, ***A Behavioral Representation for Nyquist Rate A/D Converters***, Proc. IEEE ICCAD, pp. 386-389, 1991

[MaA88] M.J.Marlett, J.A.Abraham, ***DC-IATP: An Iterative Analog circuit Test generation Program for generating DC single pattern tests,*** Proc. IEEE International Test Conference, pp. 839-844, 1988

[MaA90] H.A.Mantooth, P.E.Allen, ***Behavioral Simulation of a 3-bit Flash ADC***, Proc. IEEE International Symposium on Circuits & Systems, pp. 1356-1359, 1990

[MaA93] H.A.Mantooth, P.E.Allen, ***A Higher Level Modeling Procedure for Analog Integrated Circuits***, Analog Integrated Circuits and Signal Processing 3, 181-195, Kluwer Academic Publishers, 1993

[MaP93] S.Manetti, M.C.Piccirilli, ***Symbolic Simulators for the Fault Diagnosis of Nonlinear Circuits***, Analog Integrated Circuits & Signal Processing, Kluwer Academic Publishers, Boston, 1993

[Mas90] ***MAST - Reference manual***, Release 3.0, Analogy, Inc., October 1990

[Mas93] ***MAST - Guide to Wrting Templates***, Release 3.3c, Analogy, Inc., August 1993

[Mei93] Anne Meixner, ***Analog Fault Models for Mixed Integrated Circuit Testing***, Carnegie Mellon, Research Report No. CMUCAD-93-70, 1993

[MiS90] L.Milor, A.Sangiovanni-Vincentelli, ***Optimal test set design for analog circuits,*** Proc. IEEE ICCAD, pp. 294-297, 1990

[MLN93] J.P.Morin, F.Lémery, E.Nercessian, V.Sharma, J.Benkoski, D.Samani, ***A practical approach to Top/Down analog circuit design***, Proc. ESSCIRC'93, pp. 49-52, 1993

[MSS92] V.M.Ma, J.Singh, R.Saleh, ***Modeling, Simulation and Optimization of Analog Macromodels***, Proc. IEEE Custom Integrated Circuits Conference, pp. 12.1.1-12.1.4, 1992

[Nag75] L.W.Nagel, ***SPICE2: A Computer program to simulate circuits***, Memorandum No. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, 1975

[NCA93] N.Nagi, A.Chatterjee, J.A.Abraham, ***DRAFTS: Discretized Analog Circuit Fault Simulator,*** Proc. 30th ACM/IEEE Design Automation Conference, pp. 509-514, 1993

[NHM94] P.Nussbaum, M.Hinners, L.Menevaut, ***SimBoy: an analog simulateor interface for automated datasheet extraction***, The European Design and Test Conference, User Forum, pp. 37-41, 1994

[Ops94] ***Opsim User's Manual***, ANACAD, Revision, August 1994

[PiR90] L.T.Pillage, R.A.Rohrer, ***Asymptotic Waveform Evaluation for timing Analysis***, IEEE Transactions on Computer-Aided Design, Vol. 9, No. 4, April 1990

[Plu92] ***PLUTO User's Manual***, Release 2.0, SGS-Thomson Microelectronics, Central R&D, December 1992

[Rag89] R.Raghuram, ***Computer Simulation of Electronic Circuits***, ISBN 0-470-21331-0 John Wiley & Sons Inc, 1989

[RLR95] B.Romanowicz, P.Lerch, P.Renaud, ***Modelling and Simulation of Microsystems: An Experience with HDL-A***, Workshop on Libraries, Component Modelling and Quality Assurance, Nantes, France, in Proc. pp.167-177, April 1995

[RRP93] V.Raghavan,R.Rohrer,L.Pillage,J.Lee,J.Bracken,M.Alaybeyi, ***AWE-Inspired***, Proc. IEEE Custom Integrated Circuits Conference, pp. 18.1.1-18.1.8, 1993

[Rua91] G.Ruan, ***A Behavioral model of A/D Converters using a mixed-mode simulator***, IEEE Journal of Solid-State Circuits, Vol. 26, No. 3, March 1991

[TiS91] U.Tietze, Ch.Schenk, Electronics Circuits, Design and Applications, Springer-Verlag Berlin, Heidelberg 1991

[Sab87] ***Saber Reference Manual***, Release 3.1a, Analogy Inc., February 1987

[SaF93] P.Saramito, G.Fourneris, ***ZEBU: A multi-target Translator Environment***, SGS-Thomson Microelectronics Central R&D, Internal Documentation, September 1993

[Sim94] ***SimPilot User's Manual***, Revision 2.0, ANACAD, October 1994

[SiS91] J.Singh, R.Saleh, ***iMACSIM: A Program for Multi-level Analog Circuit Simulation***, Proc. IEEE ICCAD, pp 16-19, November 1991

[SlK92] M.Slamani, B.Kaminska, ***Analog circuit fault diagnosis based on sensitivity computation and functional testing***, IEEE Design & Test of Computers, pp 30-39, March 1992

[Spe95] ***Spectre Reference Manual*** 4.3.4, Cadence Design Systems Inc., June 1995

[SRC94] R.A.Saleh, D.Rhodes, E.Christen, B.A.A.Antao, ***Analog Harware Description Languages***, Proc. IEEE Custom Integrated Circuits Conference, pp. 15.1.1-12.1.8, 1994

[STS92] ***ST-SPICE User's Manual***, Version 3.0, SGS-Thomson Microelectronics, Central R&D, June 1992

[TiS91] U.Tietze, Ch.Schenk, ***Electronics Circuits Design and Applications,*** Springer-Verlag, Berlin, Heidelberg 1991

[TRG93] H.El Tahawy, D.Rodriguez, S.Garcia-Sabiro, J.J.Mayol, ***VHDeLDO: A new mixed mode simulation***, Proc. IEEE International Conference on Computer-Aided Design, pp 546-551, 1993

[Vac93] A.Vachoux, ***VHDL-A Rationale***, Version 2.0, Aug 1993

[Ver95] ***Verilog-XL Reference Manual*** 2.2, Cadence Design Systems Inc., June 1995

[Vop90] Voperian, ***Simplified Analysis of PWM converters using the model of the PWM switch, Part 1: continuous conduction mode***, IEEE Transactions on AES, Vol 26, No.2, March 1990

[VCC88] C.Visweswariah, R.Chadha, C.F.Chen, ***Model Development and Verification for High Level Analog Blocks***, Proc. 25th ACM/IEEE Design Automation Conference, pp 376-3824, 1988

[WAL92] A.Walker, W.E.Alexander, P.K.Lala, ***Fault diagnosis in Analog circuits using element modulation***, IEEE Design & Test of Computers, pp. 19-28, March 1992

[WFG94] P.Wambacq, F.V.Fernandez, G.Gielen, W.Sansen, ***Efficient Symbolic Computation of Approximated Small-Siganl Characteristics***, Proc. IEEE Custom Integrated Circuits Conference, pp. 21.5.1-21.5.4, 1994

Annexe 1: Simulation analogique et mixte

1 Introduction

Le but de cette annexe est de présenter les divers types de simulateurs utilisés au cours de cette thèse: les simulateurs électriques tels que SPICE et Eldo et les simulateurs mixtes analogiques digitaux, tels que le système composé de deux simulateurs distincts Verilog-Eldo.

2 La simulation électrique

Le but de ce paragraphe est de présenter un aperçu des techniques utilisées pour la simulation électrique afin de mieux appréhender les problèmes de la modélisation comportementale analogique.

2.1 La mise en équation du réseau électrique

Le réseau électrique, constitué d'un ensemble de branches, peut être décrit par un système d'équations satisfaisant aux équations de Kirchhoff des courants (KCL) et des tensions (KVL), ainsi qu'aux équations des composants [Nag75]. Chaque composant, considéré comme une interconnexion de branches, exprime les relations des tensions ou courants de ses branches, en fonction des inconnues du circuit. Par exemple, dans le cas de la Figure 1, le modèle du composant amplificateur se caractérise par deux branches *a,b* et *c,d* et deux relations définissant les courants qui y circulent.

Divers algorithmes peuvent être utilisés pour exprimer le système d'équations correspondant au réseau électrique. La ***méthode nodale*** conduit à un système très simple: les variables indépendantes du système sont les tensions de noeud, calculées par rapport à un noeud de référence. Quant au système, il est composé des équations de Kirchhoff des courants

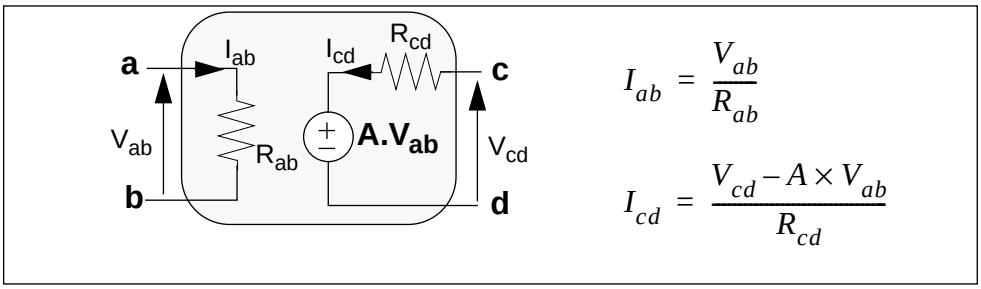


Figure 1 Modèle d'amplificateur linéaire

en chaque noeud, à l'exception du noeud de référence. Cette méthode est cependant restrictive car elle ne permet pas l'utilisation de sources de tension pour lesquelles le courant qui les traverse est une inconnue.

Il est donc nécessaire d'utiliser des méthodes hybrides, telles que l'***analyse nodale modifiée*** (Modified Nodal Analysis ou MNA): cette méthode, utilisée dans SPICE, offre un bon compromis simplicité/généralité. Les variables indépendantes du système sont constituées des tensions de noeud et des courants circulant dans les sources de tension et, de façon générale, dans tous les composants dont les équations de branche ne peuvent être représentées sous la forme explicite $i = Y \cdot v$, où Y représente l'admittance de la branche. Les équations du système sont donc constituées des équations KCL et des équations de branches pour lesquelles le courant est une inconnue.

Enfin, quelle que soit la méthode utilisée, la taille du système à résoudre dépend directement du nombre de noeuds du réseau électrique. La modélisation comportementale permet justement de réduire ce nombre, comme nous le verrons au chapitre suivant.

2.2 Calcul du point de fonctionnement (analyse DC)

Il s'agit de déterminer la valeur des tensions et courants du réseau électrique en régime permanent, c'est à dire indépendamment du temps. Le système d'équations est de façon générale non-linéaire et est résolu par des algorithmes itératifs de type ***Newton-Raphson*** ou de ***relaxation*** [Rag89].

2.2.1 Newton-Raphson

L'algorithme ***itératif*** de ***Newton-Raphson*** est utilisé dans SPICE et est bien adapté aux circuits analogiques fortement couplés.

Il est couramment employé, sous le nom de *méthode de la tangente*, pour la résolution d'équations non-linéaires à une inconnue $F(x) = 0$ et consiste, dans ce cas, à choisir pour x^{p+1}, inconnue à l'itération p+1, la racine ξ de l'application affine tangente en x^p à F: $F(x^p) + (\xi - x^p) \times F'(x^p) = 0$.

Cette méthode peut être étendue à la résolution d'un système d'équations non-linéaires en remplaçant l'équation scalaire précédente par un système linéaire où x est le vecteur inconnu et F' désigne le Jacobien de F (matrice des dérivées partielles des composantes de F). Or, dans le cas des circuits électriques, F' comporte un grand nombre de termes nuls qu'il est inutile de calculer. Ainsi, au lieu de recourir au Jacobien, chaque relation de branche du circuit est linéarisée une à une, afin de construire un nouveau réseau basé sur des schémas équivalents des éléments linéarisés.

La Figure 2 donne l'exemple du schéma équivalent linéarisé d'une diode, composant fortement non-linéaire: il est constitué d'une conductance g_{eq} et d'une source de courant i_{eq} dont les valeurs, à une itération donnée, dépendent des résultats de l'itération précédente. Signalons que le calcul de ces valeurs nécessite la définition de la dérivée partielle $\left.\frac{\partial i_d}{\partial v_d}\right|_{v^p}$, dont l'expression analytique est en fait décrite dans le modèle SPICE.

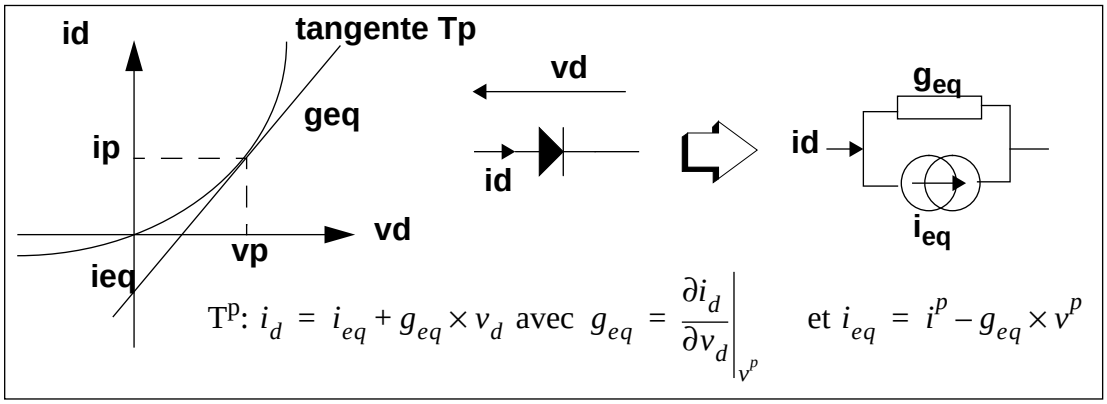


Figure 2 Modèle équivalent linéarisé d'une diode

Le nouveau système linéarisé est ensuite résolu par simple élimination de ***Gauss*** ou par factorisation triangulaire ***L.U.*** (*Lower/Upper*). Les algorithmes des *matrices creuses* sont mis en oeuvre pour obtenir la plus grande efficacité possible.

Du point de vue de la modélisation, l'inconvénient principal de cet algorithme est lié à la nécessaire définition analytique des dérivées partielles de chaque relation de branche. Cette contrainte est un réel frein à l'écriture éventuelle de modèles comportementaux.

2.2.2 Relaxation

Les algorithmes de ***relaxation***, tels que celui de ***Gauss-Seidel*** [Rag89], donnent de bons résultats pour les circuits peu couplés et spécialement pour les circuits composés de transistors MOS. Ils consistent en un découplage des variables qui sont calculées une à une en fixant les autres à leurs valeurs précédentes.

Par exemple, l'algorithme de *Gauss-Seidel* calcule les composantes du vecteur inconnu x à l'itération p+1, soit x^{p+1}, dans l'ordre des indices croissants: chaque x_i^{p+1} est solution de l'équation scalaire $f_i(x_1^{p+1}, x_2^{p+1}, \ldots, \xi, x_{i+1}^p, \ldots, x_n^p) = 0$, où f_i est la i-ème composante de la matrice *F* du système et ξ l'inconnue. On remarque que cette équation met en jeu les composantes $x_1^{p+1}, x_2^{p+1}, \ldots, x_{i-1}^{p+1}$ calculées à l'itération courante et les $x_{i+1}^p, x_{i+2}^p, \ldots, x_n^p$ calculées à l'itération précédente.

Enfin, cette équation scalaire est résolue soit:

- par un pas de la méthode de *Newton* (on parle alors de *Gauss-Seidel-Newton*):

$$x_i^{p+1} = x_i^p - \frac{f_i(x_1^{p+1}, x_2^{p+1}, \ldots, x_i^p, x_{i+1}^p, \ldots, x_n^p)}{\frac{\partial}{\partial x_i} f_i(x_1^{p+1}, x_2^{p+1}, \ldots, x_i^p, x_{i+1}^p, \ldots, x_n^p)},$$

- par la méthode du *point fixe* ou la méthode de la *sécante*. Au contraire de *Newton*, ces deux algorithmes ne nécessitent pas de dérivées partielles ce qui est intéressant pour la modélisation comportementale. Notons enfin qu'ils sont utilisés dans la boucle interne de l'algorithme de relaxation *O.S.R.* (*One Step Relaxation*) du simulateur Eldo [HeS85].

2.3 Analyse temporelle

Elle consiste en la résolution d'un système d'équations différentielles ordinaires et non-linéaires en un certain nombre de pas de temps. Là encore on distingue des méthodes itératives et des méthodes de relaxation selon le niveau de découplage des variables.

2.3.1 Méthodes itératives

Considérons tout d'abord le mécanisme itératif d'une simulation temporelle SPICE qui est décrit en Figure 3. Un système d'équations algébriques non-linéaires, dites ***discrétisées***, est élaboré à chaque pas de temps, en fonction des valeurs précédemment calculées et en utilisant

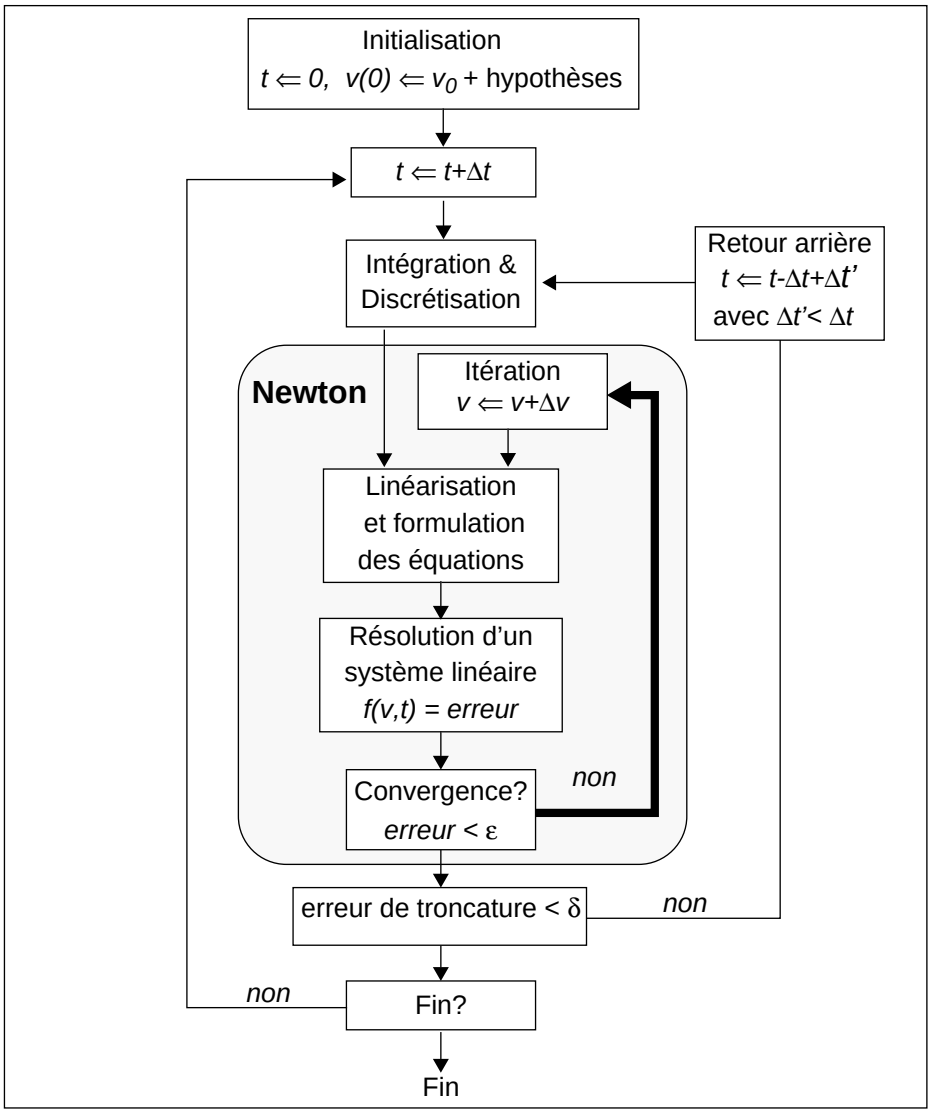


Figure 3 Schéma d'une simulation temporelle SPICE [Nag75] [Spe95]

des ***algorithmes d'intégration.*** La largeur des pas de temps est déterminée dynamiquement par le simulateur en fonction de critères de précision: si, à un pas de temps donné, l'erreur de la solution est inférieure à une certaine valeur, le simulateur tente d'augmenter le prochain pas de temps Δt, afin de réduire la durée de la simulation. Si les critères de convergence ne sont pas vérifiés, le pas de temps qui avait été initialement choisi est réduit et une nouvelle itération est effectuée.

Les ***algorithmes d'intégration*** sont définis par leur précision et stabilité [Nag75] et se classent en deux catégories: intégration explicite, nommée *Euler-directe* (*forward-Euler*) et qui est particulièrement instable, ou implicite (*backward-Euler*, *Gear*, intégration *trapézoïdale*).

Prenons l'exemple du schéma d'intégration de *backward-Euler*. Sa formule est obtenue par les développements de Taylor de x_{n+1} (valeur de x à l'instant t_{n+1}) et de sa dérivée

$\frac{dx_{n+1}}{dt}$ en fonction des grandeurs calculées à l'instant précédent t_n: x_n et $\frac{dx_n}{dt}$. Elle s'écrit:

$x_{n+1} = x_n + h_n \cdot \frac{dx_{n+1}}{dt} - \frac{h_n^2}{2} \cdot \frac{d^2 x_n}{dt^2}(\xi)$ où h_n correspond au pas de temps du simulateur entre t_n et t_{n+1} et où le terme du second ordre constitue l'*erreur de troncature* qui caractérise la précision de la méthode. Cette relation permet ainsi de déterminer $\frac{dx_{n+1}}{dt}$ en fonction de x_{n+1} et x_n (de façon approchée).

Par exemple, l'équation $i = C \cdot \frac{dv}{dt}$ d'une capacité est transformée selon ce schéma en l'équation discrétisée $i_{n+1} = \frac{C}{h_n} \cdot (v_{n+1} - v_n)$, à l'instant t_{n+1}.

2.3.2 Méthodes de relaxation

Concernant les méthodes de relaxation, le découplage des variables peut être effectué soit au niveau du système d'équations non-linéaires soit au niveau supérieur du système d'équations différentielles.

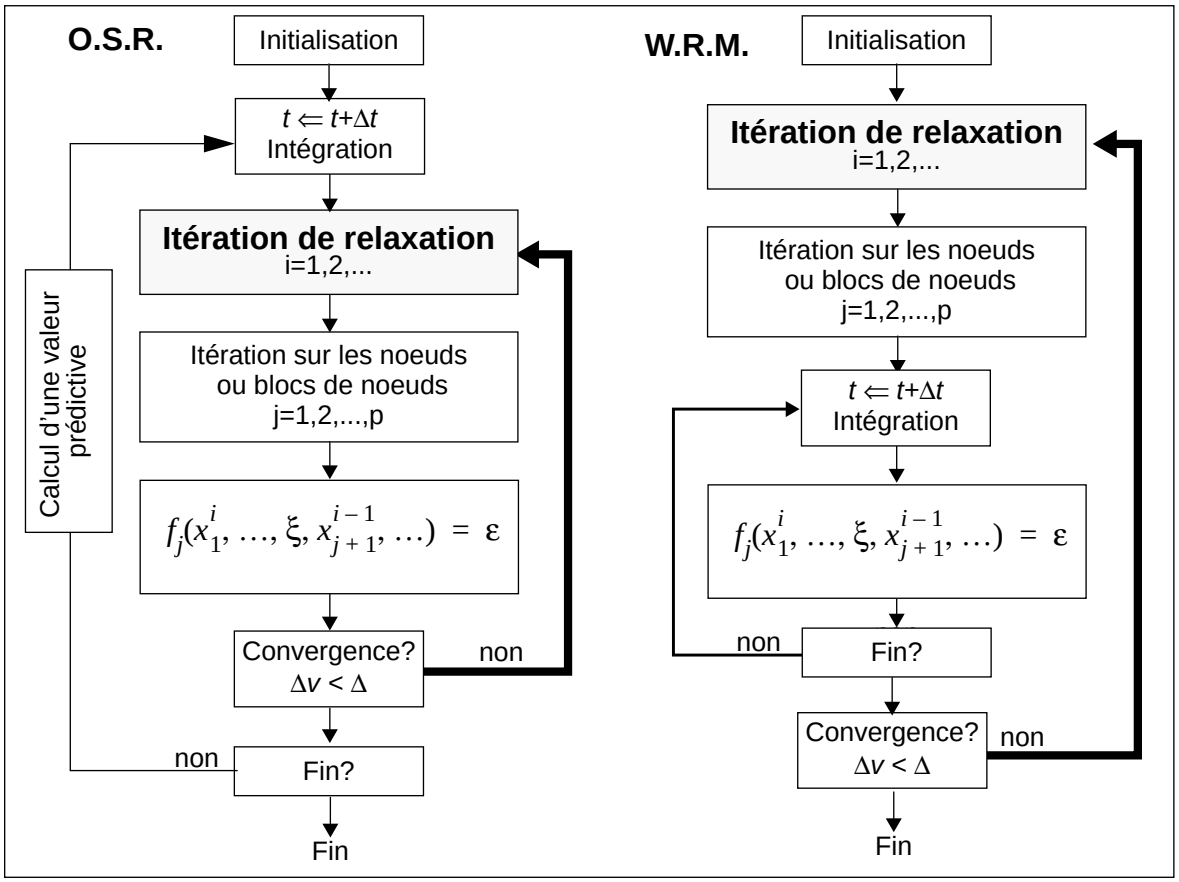


Figure 4 Algorithmes de relaxation: O.S.R. et W.R.M.

Le premier cas correspond à l'algorithme de relaxation ***O.S.R.*** de Eldo qui est présenté en Figure 4 [HeS85]. L'ordre d'imbrication des boucles est: ***temps, relaxation, noeuds***. Une

valeur prédictive est calculée à la fin de chaque pas de temps comme condition initiale pour le suivant.

Le second cas constitue l'algorithme ***W.R.M.*** ou ***Waveform Relaxation Method***, utilisée par le simulateur *Relax* [HeS85] (cf Figure 4). Ici, l'ordre d'imbrication des boucles est: ***relaxation, noeuds, temps***. Chaque variable est une fonction du temps $x_j(t)$, *définie à chaque pas de temps de la simulation, et qui est déterminée* à partir des autres variables elles-aussi fonctions du temps. Par rapport au schéma précédent, la boucle de relaxation est simplement déplacée à un niveau supérieur. Cependant, ce type de méthode requiert la mémorisation de toutes les fonctions $x_j(t)$ et en pratique, la simulation est décomposée en plusieurs fenêtres temporelles successives.

2.4 Analyse fréquentielle ou AC

L'analyse AC, effectuée par un simulateur de type SPICE, correspond à une ***étude linéaire*** de la réponse fréquentielle du circuit pour des sources sinusoïdales de même fréquence, mais de phases éventuellement différentes. Chaque élément non-linéaire est tout d'abord remplacé par un modèle équivalent *linéarisé*, dont les valeurs sont déterminées par une analyse initiale du point de fonctionnement. Le système d'équations différentielles est alors transformé en un système d'équations algébriques appartenant au domaine complexe par transformation de Fourier et en effectuant les substitutions $\frac{dv}{dt} \Rightarrow j\omega V$, où ω est la pulsation des stimuli en rad/s. Les termes réactifs, dépendant de ω, sont ensuite évalués aux diverses fréquences de l'analyse.

3 La simulation temporelle analogique-digitale

Seront ici traités les problèmes de communication entre les deux simulateurs analogiques et digitaux, c'est à dire la conversion des données et la synchronisation.

3.1 La conversion A/D et D/A

Des convertisseurs fictifs A/D et D/A doivent être introduits dans le schéma au niveau

des noeuds d'interface entre les blocs analogiques et les blocs digitaux. Différents modèles de convertisseurs sont utilisés en fonction des types de données et du niveau de précision souhaitée.

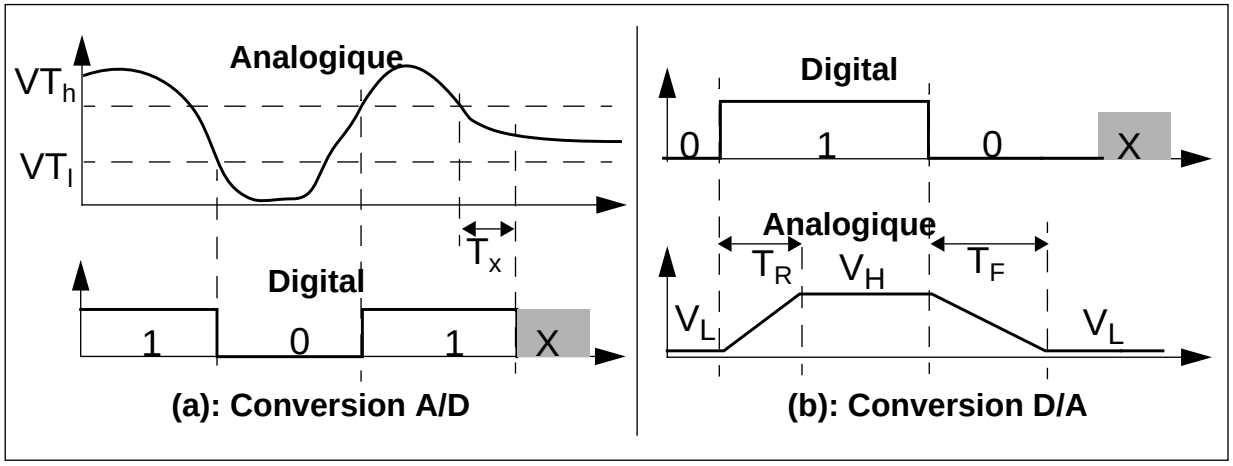


Figure 5 Conversion analogique/digitale et digitale/analogique

Le principe de la ***conversion A/D*** est exposée en Figure 5a et se caractérise par un seuil haut VTh, un seuil bas VTl et un temps Tx à partir duquel le signal logique passe à l'état X lorsque le signal analogique se trouve entre les deux seuils. Le modèle du convertisseur A/D est décrit comme un comparateur à hystérésis, qui peut éventuellement posséder une impédance d'entrée.

La ***conversion D/A*** (cf Figure 5b), basée par exemple sur une source de tension contrôlée par un signal digital (cf Figure 6a), génère un signal analogique dont les paramètres sont les temps de montée TR et de descente TF, ainsi que les tensions hautes VH et basses VL. Dans ce simple modèle, l'état indéterminé X est ignoré et l'état haute impédance Z peut correspondre à la tension moyenne *(VH+VL)/2*. De plus, il doit être complété d'une résistance de sortie afin de connecter plusieurs signaux en un même noeud (cf Figure 6b). Enfin, il est aussi possible d'utiliser une structure CMOS [AAS95] qui permet en particulier de modéliser correctement l'état haute impédance (cf Figure 6c). Notons que les MOS peuvent être remplacés par des modèles d'interrupteurs résistifs.

Le modèle IRC de la Figure 7 est proposé par Eldo [Eld94] pour des signaux digitaux de type VHDL *STD_LOGIC_1164*, qui sont caractérisés par la notion supplémentaire de *force*: par exemple, le niveau haut *fort* noté '1' est distinct du niveau haut *faible* noté 'H'. Les valeurs I, R et C dépendent donc de l'état et de la force de la variable digitale.

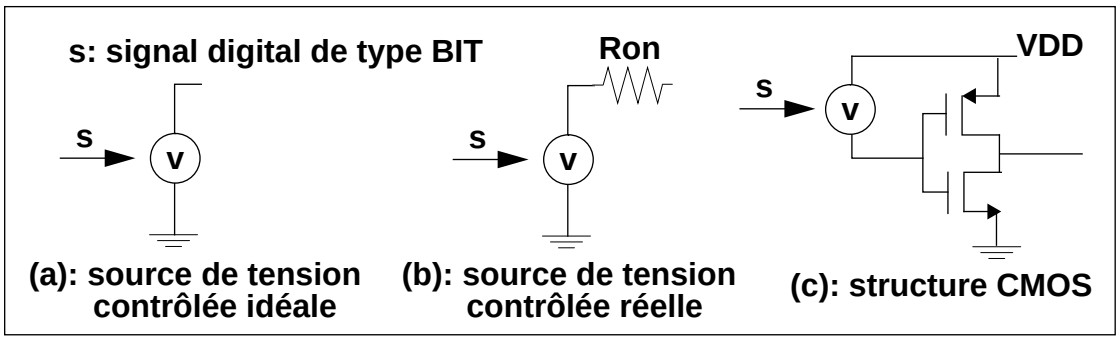


Figure 6 Convertisseurs D/A contrôlés par un signal digital de type Bit

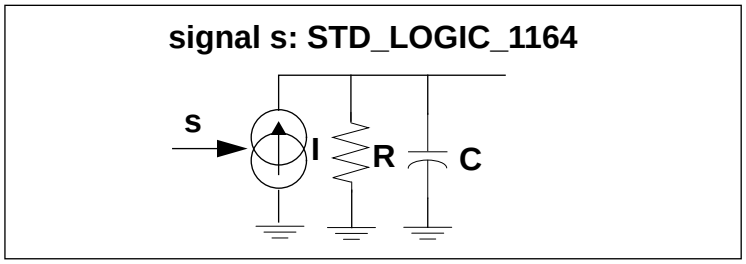


Figure 7 Convertisseurs D/A tenant compte de la *force*

3.2 La synchronisation

On distingue deux catégories d'algorithmes [TRG93] qui se complètent et qui sont en pratique combinés (cf Figure 8):

- l'algorithme du ***pas bloqué*** ou ***lock-step***: selon cette méthode, les pas de temps du simulateur analogique, **A**, sont ajustés en fonction de ceux du simulateur digital, **D**. Cela signifie que **D** impose une évaluation de **A** lors de chaque évènement digital (par exemple au point **I** de la Figure 8a). Lors d'un évènement analogique/digital (point **II**), la liste d'évènements de **D** est réévaluée. Cependant, cet algorithme est pénalisant car les pas de temps de **A** sont "bloqués" par ceux de **D** même si les critères de précision permettent théoriquement à **A** de les augmenter (point **III**).
- l'algorithme avec ***retour en arrière*** ou ***back-tracking*** du simulateur analogique (cf Figure 8b): il résout le problème précédent car **A** peut avancer dans le temps aussi loin que la précision le lui permet. Lorsqu'un évènement mixte digital /analogique intervient (point **IV**), **A** revient en arrière pour effectuer un nouveau calcul et rajuste son pas de temps. **D** prend en compte les évènements analogiques/digitaux comme précédemment.

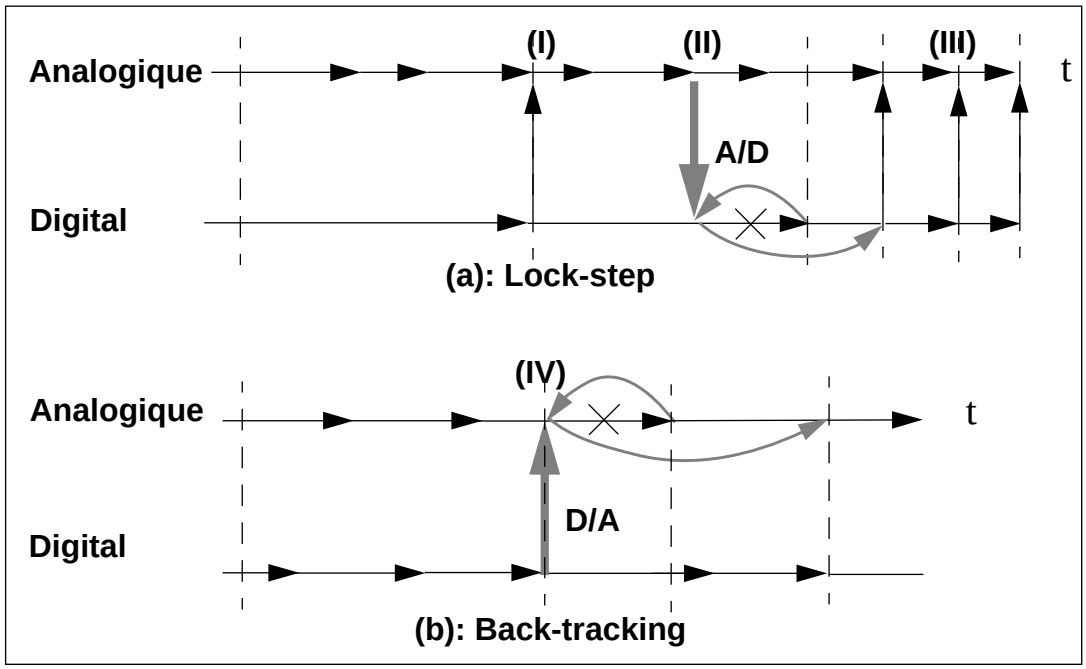


Figure 8 Méthodes de synchronisation

Annexe 2: Modélisation pluri-disciplinaire

1 Introduction

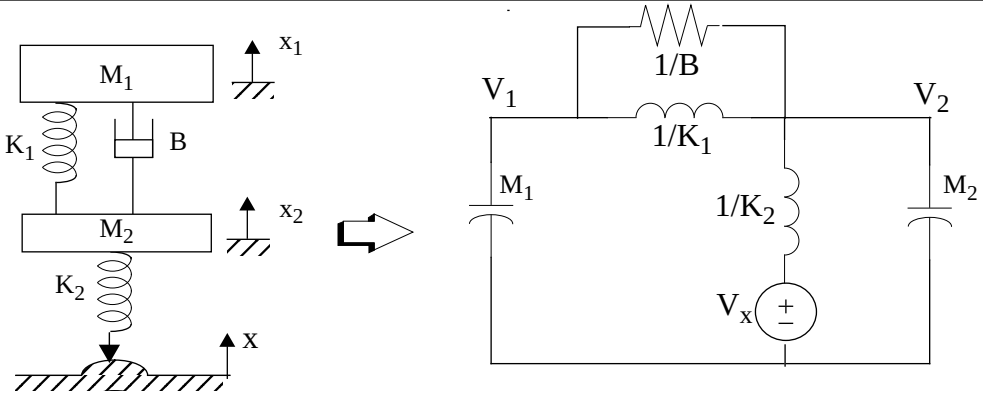
Cette annexe donne quelques exemples de systèmes pluri-disciplinaires: mécaniques, thermiques et hydrauliques. Le Tableau 1 illustre les “équivalences” qui peuvent exister entre des composants appartenant à des domaines physiques distincts.

Tableau 1 Analogies entre différents composants physiques

Forme générique → (π, φ)	$\pi(t) = a \cdot \varphi(t)$	$b \cdot \frac{d\pi}{dt} = \varphi(t)$	$\pi(t) = c \cdot \frac{d\varphi}{dt}$
Électrique (v, i)	Résistance R(Ω): $v = R \cdot i$	Capacité C(F): $C \cdot \frac{dv}{dt} = i$	Inductance L(H): $v = L \cdot \frac{di}{dt}$
Mécanique Translation (v, F)	Amortissement B: $v = F/B$	Masse M (kg): $M \cdot \frac{dv}{dt} = F$	Ressort de raideur K: $v = \frac{1}{K} \cdot \frac{dF}{dt}$
Thermique (T, Φ)	Résistance thermique équivalente Rt (K.s/J): $T_1 - T_2 = R_t \cdot \Phi$	Capacité thermique m.C (J/K): $m \cdot C \cdot \frac{dT}{dt} = \Phi$	
Hydraulique (P, Qv) ou (z, Qv)	Perte de charge dans les conduites et aux singularités (bifurcations, rétrécissements, élargissements...)	Réservoir de surface A, hauteur z: $A \cdot \frac{dz}{dt} = Q_v$	
Pneumatique (P, Q_m) (gaz parfaits)		Réservoir de capacité m_0 à P_0 $\frac{m_0}{P_0} \cdot \frac{dP}{dt} = Q_m$	

2 Système mécanique

Ainsi, dans le cas d'un système ***mécanique***, on peut effectuer une analogie entre force et courant, la loi de Kirchhoff des courants étant remplacée par la relation fondamentale de la dynamique appliquée à chaque élément massique du système. Le Tableau 1 établit les correspondances entre certains composants mécaniques et électriques, en utilisant la vitesse comme variable de type potentiel. Notons que pour obtenir un système d'équations différentielles ordinaires, tout système mécanique élastique devra être modélisé à l'aide d'un ensemble discret de masses, reliées entre elles par des éléments de type ressort ou amortisseur sans masse. La Figure 1 présente un exemple simple de système mécanique (roue d'une voiture) et le modèle électrique correspondant où x_1, x_2 désignent les déplacements des masses M_1 (carrosserie) et M_2 (roue) par rapport à leurs positions d'équilibre et V_1, V_2 sont leurs vitesses respectives. Le profil de la route est représenté par une source de tension V_x dont la valeur est la dérivée de l'écart de hauteur x.



Système à deux degrés de liberté, v1 et v2:

$$(1)\ M_1 \cdot \frac{dv_1}{dt} + K_1 \cdot \int (v_1 - v_2)\, dt + B \cdot (v_1 - v_2) = 0$$

$$(2)\ M_2 \cdot \frac{dv_2}{dt} + K_1 \cdot \int (v_2 - v_1)\, dt + B \cdot (v_2 - v_1) + K_2 \cdot \int (v_2 - v_x)\, dt = 0$$

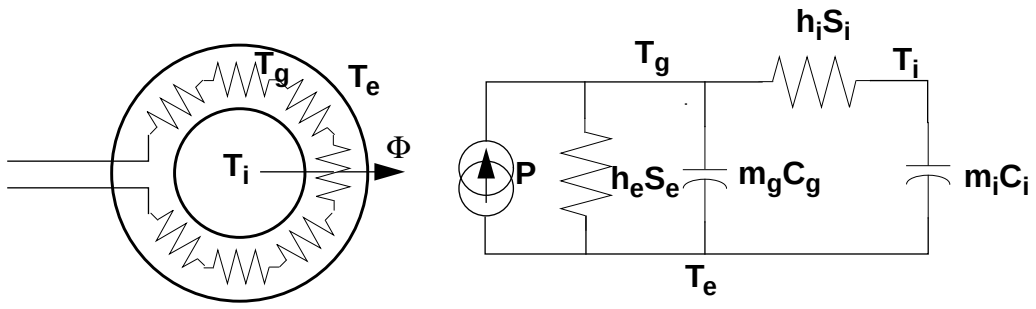
Figure 1 Système mécanique et son équivalent électrique [ARN70]

Notons que le système d'équations pourrait être aussi décrit à l'aide du couple position-force, au lieu du couple vitesse-force. Dans ce cas, un ressort serait modélisé par une résistance, un amortisseur par une capacité et la force d'inertie nécessiterait par contre un nouveau composant effectuant une dérivation seconde.

3 Système thermique

À l'intérieur des milieux matériels, plusieurs possibilités d'échange de chaleur existent entre deux points du milieu, à savoir la conduction, la convection et le rayonnement [JEA89]. Dans une approche "industrielle", le système thermique est décomposé en plusieurs sous-systèmes associés à chaque phénomène et décrit la conservation de l'énergie. En présence de convection, on utilise la loi classique de Newton $\Phi = h \cdot S \cdot (T_{paroi} - T_{fluide})$, où h désigne un coefficient d'échange superficiel, S la surface d'échange et Φ le flux de chaleur. Pour les échanges de conduction en régime permanent, la notion de résistance thermique R_t est employée. Elle est définie par la relation $\Phi_{12} = (T_1 - T_2) / R_t$ où Φ_{12} est le flux circulant entre les deux isothermes T_1 et T_2 . Enfin, le stockage de la chaleur est décrit par des termes dérivées $m \cdot C \cdot \frac{\partial T}{\partial t} = \Phi$, où m est la masse du corps considéré et C la chaleur spécifique.

La Figure 2 présente l'exemple d'un four électrique constitué d'une gaine cylindrique chauffée électriquement [ARN70], où *Se* et *Si* sont les surfaces externes et internes de la gaine, *he* et *hi* les coefficients d'échange superficiel des surfaces *Se* et *Si*, *mgCg* et *miCi* les capacités thermiques de la gaine et du milieu interne, et enfin *P* désigne la puissance électrique reçue par la gaine. On suppose d'une manière simplifiée que les températures du milieu externe T_e, de la gaine T_g et du milieu interne T_i sont uniformes (c'est à dire que les résistances thermiques y sont négligeables). La première relation de la Figure 2 est relative au milieu intérieur et la seconde à la gaine.



Système à deux variables indépendantes: Tg et Ti

(1) Milieu intérieur: $m_i \cdot C_i \cdot \frac{dT_i}{dt} = h_i \cdot S_i \cdot (T_g - T_i)$

(2) Gaine: $m_g \cdot C_g \cdot \frac{dT_g}{dt} = h_i \cdot S_i \cdot (T_i - T_g) + h_e \cdot S_e \cdot (T_e - T_g) + P$

Figure 2 Représentation simplifiée d'un four électrique [ARN70]

4 Système hydraulique

Dans le cas du transfert des fluides parfaits, la conservation de l'énergie mécanique est exprimée par la relation de Bernouilli appliquée le long d'une ligne de courant: l'expression $P + \rho \cdot g \cdot Z + \frac{1}{2} \cdot \rho \cdot V^2$, désignant la charge H, est une constante (P est la pression, ρ la masse volumique, g l'accélération de pesanteur, Z le déplacement vertical, V la vitesse de débit). En réalité, on observe des pertes de charge dans les conduites, dues entre autres à la viscosité des fluides. Ainsi, pour un liquide, cette perte est exprimée par la relation $\Delta H = \psi \cdot \frac{L}{d} \cdot \frac{1}{2} \cdot \rho \cdot V^2 = \alpha \cdot Q^2$ où ψ est un coefficient sans dimension, L la longueur de la conduite, d son diamètre et V la vitesse équivalente à travers une section droite S, telle que le débit soit $Q = S \cdot V$ [JEA89]. Elle est donc proportionnelle au carré du débit.

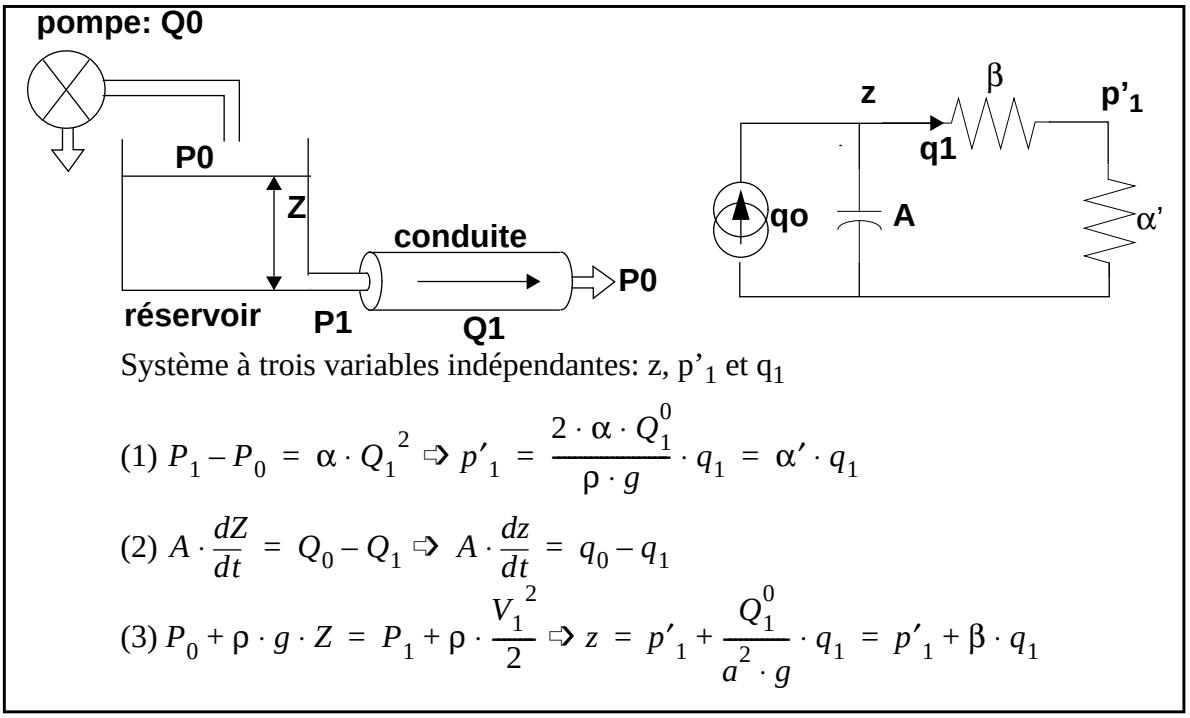


Figure 3 Représentation d'un réseau hydraulique

La Figure 3 présente un exemple simple de système hydraulique constitué d'une pompe de débit Q_0 qui alimente un réservoir de hauteur d'eau *Z* et de surface *A*. Ce dernier se vide dans une canalisation cylindrique de section *a* et dont le coefficient de perte de charge est α. Les variables indépendantes de ce système sont la pression P_1, la hauteur d'eau *Z* et le débit Q_1 circulant dans la conduite. La première relation de la Figure 3 définit la perte de charge dans la conduite, la deuxième la conservation de la matière dans le réservoir et la troisième la relation de Bernouilli appliquée entre la surface et la sortie du réservoir. Dans le cas où on étudie la

régulation de la hauteur d'eau du réservoir pour obtenir tel débit en sortie, on peut linéariser les équations autour d'une valeur moyenne, en prenant $Z = Z^0 + z$, $P = P^0 + p$ et $Q = Q^0 + q$. Le schéma électrique équivalent permet d'étudier la fonction de transfert du système. Notons que les variables de type "potentiel" qui sont utilisées ont la dimension d'une hauteur: z et $p' = p / (\rho \cdot g)$.

Annexe 3: Système d'*AIR-BAG*

Publication "***Behavioral Models for complex top-down analog/digital system simulation***"
soutenue à la conférence "***European Simulation Multi-conference 1994***",
Universitat Politecnica de Catalunya, Barcelona, Spain, June 1-3, 1994
in Proceedings "Modelling and Simulation ESM 94", pp 1050-1054

BEHAVIORAL MODELS FOR COMPLEX TOP/DOWN ANALOG/DIGITAL SYSTEM SIMULATION†

F.Lémery*, J.P.Morin, E.Nercessian, J. Benkoski, D.Samani
SGS-Thomson Microelectronics
Ave. des Martyrs - BP 217 - 38019 Grenoble cedex - France
* jointly with INPG/TIMA
46, av Félix Vialet - 38031 Grenoble cedex - France

ABSTRACT

With the increase of the complex mixed analog/digital circuit market, it is necessary to address the problems of their simulation and modeling. The top/down design methodology, which is based on behavioral modeling, has been successfully applied to complex digital circuits and more recently to analog ones. The aim of this paper is to apply the same methodology for simulation of analog/digital systems, taking as an example a security air-bag system, which requires accurate evaluation of its global performance.

A study of both digital and analog behavioral modeling, at different abstraction levels, is presented and relevant problems of mixed analog/digital modeling are pointed out. After a short description of the air-bag system, behavioral modeling and structural description of some analog and digital functional modules are detailed. Finally, we provide some mixed simulation results showing the most important signals, together with a presentation of the mixed CAD environment used to obtain them.

1 INTRODUCTION

With the advent of new integrated circuit technologies, there is no limitation for the designers to integrate onto a single chip a complete system including both analog and digital functions. Applications in the telecommunication, signal processing, automotive and all process control fields are numerous. Such circuits require analog interfaces (sensors and operators), A/D and D/A converters, analog or digital data processors (filters) and some control logic. Moreover, with the increase in frequencies, more analog components are needed.

Design of such complex analog/digital systems represents a real bottleneck. Indeed, the major parts of design cycle of digital circuits have been automated through the efficient top/down design methodology but many analog circuits are still designed manually.

During the top/down digital design methodology, behavioral models, that describe a module function without giving any information about how this function is performed, are written at each abstraction level. A synthesis step is then performed, which breaks down the models into interconnected sub-models, to get a structural representation. Different architectures can be studied while verifying their functionalities through simulations in order to meet the current level specifications.

Many attempts have been made to apply the same approach for analog circuits, especially for the analog part of A/D converters (Liu et al. 1991; Gielen et al. 1992) or for complex phase-locked-loop (PLL) circuit (Antao et al. 1993; Morin et al. 1993). Top/down methodologies for mixed systems have also been theoretically described (Liu et al. 1991) but it seems that mixed systems more complex than A/D converters or PLLs have not been addressed.

Our contribution in this paper is a presentation of what mixed behavioral system modeling involves, and how modeling of such complex mixed system as an air-bag system (Brambilla 1982) can be achieved.

2 MIXED BEHAVIORAL LEVELS

Behavioral modeling has long been applied to the design and synthesis of digital circuits where different abstraction levels are defined (Gajski et al. 1992): system, register transfer logic (RTL), logic and gates (Table 1).

Table 1 Abstraction levels for Digital systems

Level Name	Behavioral Representation	Structural Representation
System	Algorithms Truth-Tables	Processors Memories
R.T.L	Register transfers	Registers ALUs
Logic	Boolean Equations	Gates
Transistor	Transfer Function Timing diagrams	Transistors (Analog domain)

In the analog domain, we can consider the system level, the functional level which represents all the analog functionalities, and the component level (Table 2). Here, the number of description levels depend highly on the complexity of the global system and a mix of different levels is much more required.

Table 2 Abstraction levels for Analog systems

Level Name	Behavioral Representation	Structural Representation
System	Algorithms, Equations	PLLs, Filters, OpAmps Comparators Math. Modules
Functional	Equations, s Laplace or z transfer functions, Macromodels	R,L,C, Transistors
Component	Electrical Equations	Layout (Physical domain)

We must note that analog behavioral modeling is carried out using either “macro-models” (Connelly and Choi 1992), that are built using only SPICE primitives, or dedicated hardware description languages (MAST 1989; FAS 1992), that provide the basic

† This work was partly supported by the Joint European Submicron Silicon (JESSI) AC3: HDL Component Modeling Libraries

functionalities of any high-level programming languages plus specific capabilities such as a model interface with the network, different analysis modes and description of non-linear differential equations.

In practice, behavioral modeling of mixed systems depends highly on the simulator implementations (Tahawy et al. 1993). Digital simulators are based on event-driven and signal strength resolution algorithms while electrical simulators manipulate a set of non-linear differential equations solved by iteration techniques such as Newton-Raphson or relaxation. Synchronization algorithms that schedule which of the analog or digital engine runs and data conversion between the two engines are very important topics. From a user point of view, the different modeling configurations are the following:

- a single simulator, composed of an analog and a digital algorithm, and a single hardware description language (HDL). This can result from the extension of an analog simulator in order to handle digital features or the reverse. For example, event-driven and time scheduling features are implemented in some analog simulators (MAST 1989).
- two separate analog and digital simulators, that communicate over a backplane, and two independent description languages. In this case, it is not possible to describe behaviorally an ally mixed circuit such as an A/D or a D/A converter, with a mix of analog and digital inputs/outputs, without block partitioning.
- separate analog and digital simulators and a single analog/digital description language. Analog and digital statements have to be gathered into two separate sets.

Generally, today's modeling and simulation of mixed circuits use the first and second configurations. We must note that a single HDL is of course the best solution in order to address mixed behavioral modeling.

3 MIXED SYSTEM APPLICATION

Our work is based on the system specifications given in (Moser 1993). The architecture of an air-bag system is presented in Fig.1.

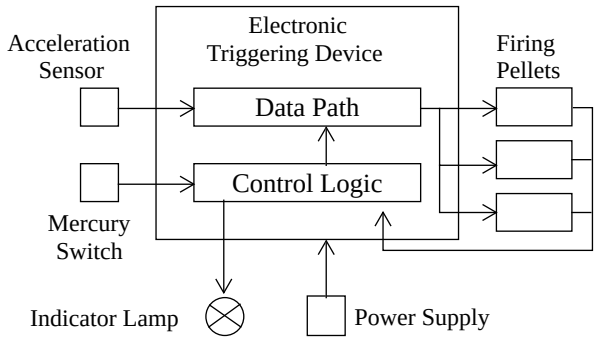


Figure 1: Air-Bag System

The goal of the electronic triggering device is to process the acceleration sensor and mercury-switch signals in order to trigger the firing pellets (one for the air-bag of the driver and two for the seat-belt tensioners of the front-seat passengers) at the proper time and only in case of accident exceeding a certain speed. A check of the validity of internal signals is also performed. The data-processing corresponds to the analog part and the control-logic to the digital one.

The system involves also some interfaces between mechanical and electrical domains such as an acceleration sensor, a mercury-switch, which is closed only if the deceleration is greater than 1.7g, and three firing pellets.

The work presented hereafter concerns the behavioral modeling of the acceleration sensor, the mercury-switch, the datapath, the three firing-pellets and a part of the control-logic block (the self-test unit and circuit monitors). The purpose of this modeling task is to verify the function of the specified system through simulations. The circuit schematic at the highest abstraction level that we have considered, is presented in Fig. 2. The functional blocks have all been described in a behavioral manner and many of them have also a structural description.

The behavioral modeling phase has been performed with the analog behavioral modeling language FAS (FAS 1992) associated to the electrical simulator ELDO (Eldo 1992). The logic blocks were described by using boolean equations and special functions that can generate voltage edges at the outputs within a certain delay and switching time.

The next step has involved a description of the architecture of some functional blocks. The impact of the architectural choices on the whole system performance has been analyzed through analog or mixed analog/digital simulations. The different blocks have been decomposed into generic parameterized macromodels and logic gates coming from an ideal functional technology-independent library and other dedicated analog behavioral models. In this way, the architectures of the acceleration sensor, the mercury-switch, the data-path, the firing-pellets, the monitors and the control-logic block were studied.

Let us now consider some blocks such as the acceleration sensor, the acceleration processing data-path and the control-logic block in order to explain how they have been modeled.

3.1 The Acceleration Sensor Model

The acceleration sensor generates a voltage proportional to the acceleration. An internal multiplexer switches between the previous signal and a self-test signal (*test*) which is generated by a self-test unit, each time the car is started. Fig. 3 shows the structural description which is based on two SPICE macro-models, an ideal amplifier and an adder, and on a behavioral multiplexer. The behavioral model written in FAS is shown below:

```
amodel SENSOR (acc, test, u, ts) ;                (* Header *)
   declare param gain,offset,vth,tcom,td: real;(* Parameters *)
   declare pin acc, test, ts, u: electrical;      (* I/Os *)
   declare state v,vtest,vts,vu,tend: real;       (* State variables *)
initialize
   vsource(u)                                     (* Voltage source *)
endinitialize
analog
if (mode=DC) then                                 (* DC operating point *)
   make v = gain * volt.value(acc) + offset       (* Linear equation *)
   make vts = volt.value(ts)
   if (vts > vth) then                            (* Multiplexer *)
   make vu = volt.value(test)
   else make vu = v
   endif
   make volt.across(u) = vu
   make tend = 0.0
else                                              (* Time analysis *)
   make v = gain * volt.value(acc) + offset       (* Linear equation *)
   make vtest = volt.value(test)
   if (time>tend) then                            (* Multiplexer *)
      if volt.rising(ts,vth) then                 (* Event detection *)
                                                  (* Ramp generation *)
```

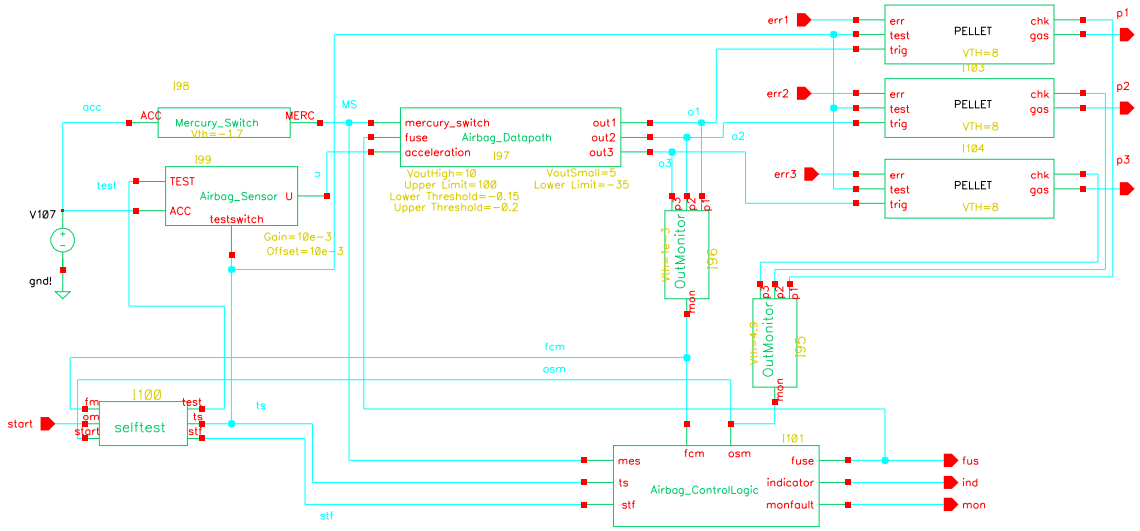


Figure 2: The System Architecture

```
      make volt.across(u)= volt.slope(vtest,tcom,td)
      make tend = time+td+tcom              (* Ramp end time *)
    else
      if volt.falling(ts,vth) then
        make volt.across(u) = volt.slope(v,tcom,td)
        make tend = time+td+tcom
      endif
    endif
  endif
endif
endanalog
endmodel
```

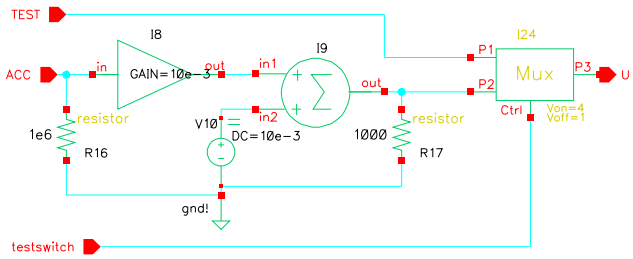


Figure 3: Sensor Architecture

3.2 The Data-Path Model

The main function of this component is to generate triggering signals for the firing-pellets at the proper time and only in case of a real accident. The sensor signal is first filtered in order to eliminate low-frequencies, then it is limited and reduced by 4g. An integration of the resulting signal, with respect to time and limited to negative values, is performed in order to generate a speed difference, which is compared to two different thresholds (one for the air-bag and one for the seat-belt tensioners). The datapath output levels (high, small, zero) generated by the three output stages, depend on these comparison results, on the mercury-switch signal, indicating an important deceleration, and on a digital signal (fuse command) coming from the control-logic block.

The structural model is presented in Fig.4 and the behavioral representation in FAS is the following:

```
amodel DATAPATH(u, fus, ms, o1, o2, o3);   (* Header *)
declare pin u, fus, ms, o1, o2, o3: electrical;  (* I/Os *)
declare param fc,a,umax,umin,offset,lth:real; (* Parameters *)
declare param uth,vhigh,vsmall,td,tcom: real ;
declare usep oo:real;                      (* Internal parameters *)
declare istate u1: real;                   (* Implicit variable *)
declare state u0,expr,u3,u4,u3old:real;    (* State variables *)
declare state next,old:real;
declare local vfus,vms,u2,utmp: real;      (* Local variable *)
initialize
   vsource(o3)                             (* Voltage source *)
   make oo=6.28*fc
   make tend=0.0
endinitialize
analog
if (mode=DC) then                          (* DC operating point *)
   make u0 = volt.value(u)
   solve(u1,u1-0.0)
   make volt.across(o3)= 0.0
   make next=0.0
endif
if (mode=TRANS) then                       (* Time analysis *)
   make u0 = volt.value(u)
                                           (* High-pass filter *)
   make expr = state.dt(u1)+oo*u1-state.dt(u0)
   solve(u1,expr)                          (* Differential equation *)
   make u2 = a*u1                          (* Gain *)
   if (u2>umax) then make u2 = umax        (* Limitation *)
   else
      if (u2<umin) then make u2 = umin endif
   endif
   make u3 = u2-offset                     (* Reduction *)
   make u3old = state.last_value(u3)
                                           (* Integration *)
   make utmp = state.last_value(u4) + (u3 + u3old)* 0.5 * tstep

   if (utmp>=0) then                       (* Limitation *)
      make u4 = 0.0
      make u3=0.0
   else make u4 = utmp
   endif
                                           (* Output stage o3 *)
   make vfus = volt.value(fus)
   make vms = volt.value(ms)
   if (time > tend) then                   (* Ramp end control *)
      if (u4 < lth) then
         if (vfus > vth) and (vms > vth) then  (* Boolean equation *)
            make next = vhigh
         else make next = vsmall
         endif
      else make next = 0.0
      endif
      make old = state.last_value(next)    (* Last time step value *)
      if abs(next-old)>1e-6 then           (* Event detection *)
         make tend = time+tcom+td          (* Ramp end time *)
         if (next > old) then              (* Ramp generation *)
            make volt.across(o3) = volt.rise(old,next,tcom,td)
   else make volt.across(o3) = volt.fall(old,next,tcom,td)
         endif
      endif
   endif (* O1 and O2 output stages are similar to O3 output stage *)
endif
endanalog
endmodel
```

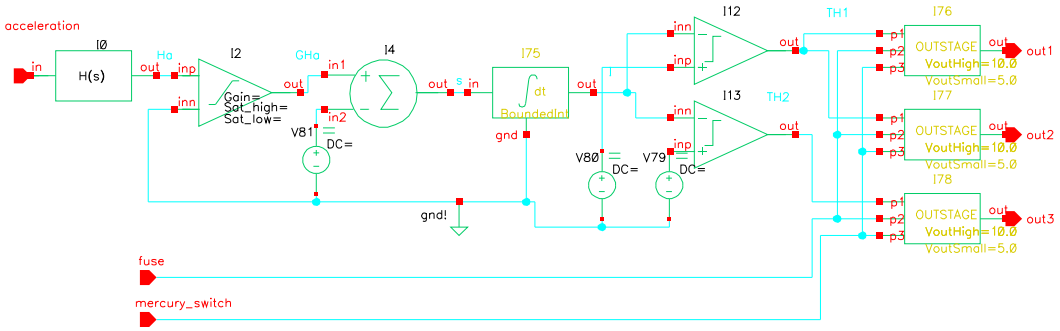


Figure 4: The Data-Path Architecture

3.3 The Control-Logic Model

This digital block was modeled in a behavioral fashion with an analog behavioral modeling language which provides boolean operators and ramp generating functions. These functions, whose parameters are delay time, rising or falling time, and output voltage levels, let the designer schedule at which time a "digital" event must occur in the future. Note that the previously described data-path model uses also such functions for the output stage.

However, such a logic module is simulated more efficiently with a digital simulator such as VERILOG (Verilog 1991). Therefore, this control-logic block has been described structurally using Verilog functional logic gates. Mixed-mode simulations of the whole analog/digital system have been thus performed.

4 MODELING AND SIMULATION ENVIRONMENT

The Analog Environment used for the modeling and the simulation of the air-bag system is based on the Cadence Design Framework II and was already described (Morin et al. 1993). Apart from the basic functionality of any front-end environment (hierarchical netlister, user interface for creating simulation commands, launching simulators and displaying results), its added features are the behavioral view management. This new feature allows the designer to easily use models in either Top/Down or Bottom/Up design methodologies.

The main characteristics of this behavioral view management are:

- Behavioral description of cells are stored in the database.
- Template of the textual description is automatically generated by the system when creating a new behavioral cell view. This feature together with the storage of the view in the database insures consistency between the different representations of the cell (i.e. schematic/behavioral).
- The switch from one description (i.e. schematic) to an other (i.e. behavioral) and vice-versa is performed, instance by instance, by a simple selection of the cell on the schematic.
- The two types of behavioral models styles are taken into account.

This unified CAD environment has been extended to allow mixed signals simulations (UNICAD 1993), based on the analog simulator ELDO (Eldo 1992), coupled with the digital one VERILOG (Verilog 1991).

This environment consists of:

- An instance-based partitioner,
- An analyser identifying the type (analog/digital) of each instances and nodes, together with the location of the interface nodes and the direction of the needed converters (A/D or D/A),
- A graphical interactive flow that guides the user through the different steps of a mixed mode run which usually appears more complex and less "intuitive" to the designer than the classical analog ones.

5 SIMULATION RESULTS

We present in Fig.5 some mixed-mode simulation results for the self-test phase and for a frontal accident. During the self-test phase, the self-test unit model generates a negative acceleration (TEST) of -20 g during 15 ms and then a positive one of 100 g during 3 ms to reset the system. In this simulation, the self-test fault signal (STF) becomes high at 13ms indicating that an error occurred. This fault arises from the seat-belt tensioner speed threshold value we have set in order to simulate a failure in the system: it causes the datapath and pellet outputs O3 and C3 to rise at 11ms, outside the time slot (13ms to 19ms) during which transitions are detected by the self-test unit.

In the last simulation phase, after 25 ms, an acceleration (G) of -100 g is simulated in order to compare control signals (C1, C2, C3) and pellet triggering indicating signals (P1, P2, P3) that are not activated in the self-test phase. Fig. 5 shows the main analog and digital signals. Note that some interface nodes between analog and digital parts are displayed in both domains (FCM and OSM for example).

This analog/digital simulation was performed using only analog behavioral models and functional gates in 3 min of CPU time on a SUN SPARCstation 1.

Fig.6 shows analog results of the structurally described system for a frontal accident for which acceleration is simulated using a piece-wise linear voltage source fitted to experimental crash measurements. The time where the pellets are triggered depends on the initial speed whose value is here 50km/h. The seat-belt tensioner is triggered at 20.9ms and the airbags are within 25ms. All these results correspond to the specified requirements.

6 CONCLUSION

This paper has demonstrated the need for multi-level and mixed analog/digital behavioral modeling, in order to simulate a complex mixed system and thus, evaluate its performance against its requirements.

Mixed behavioral modeling has been achieved through a system decomposition into pure analog and digital modules and using an extended analog description language. A unified mixed analog/digital hardware description language would certainly ease this modeling work and let us describe the system at higher abstraction levels.

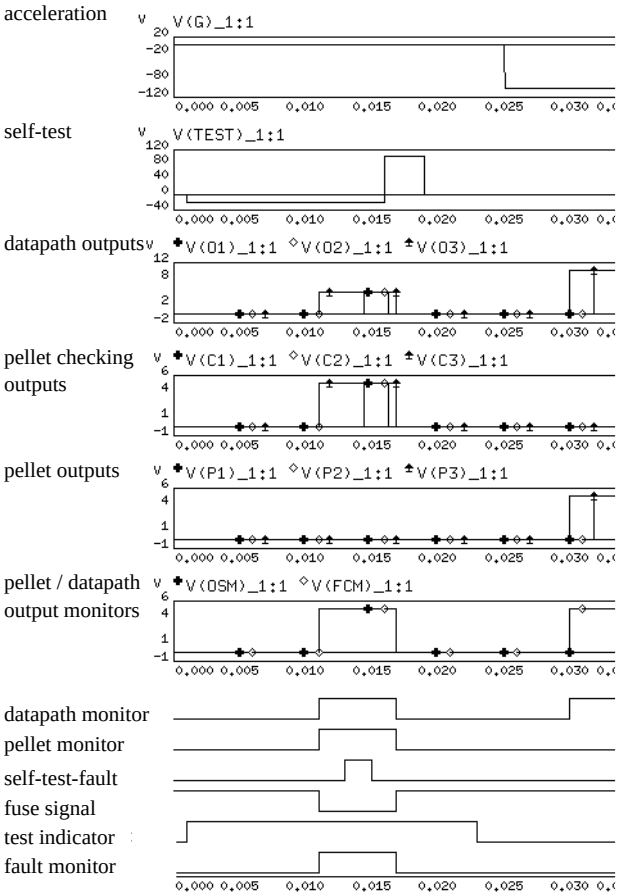


Figure 5: Self-Test Phase and Accident -100g: Mixed results

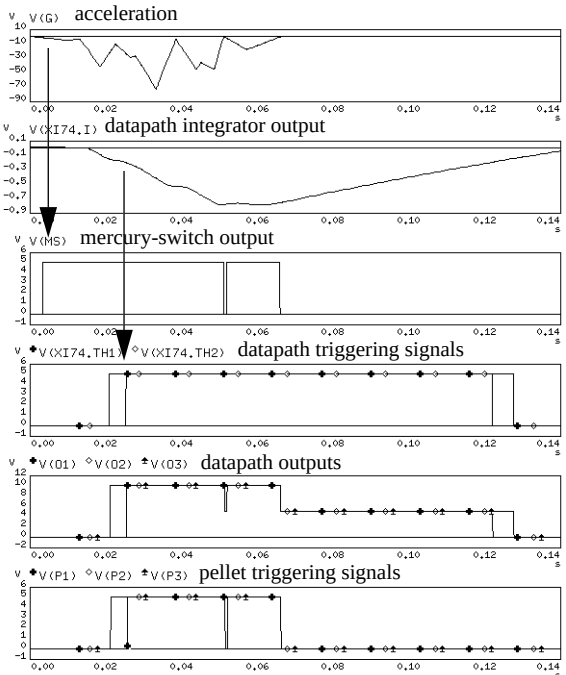


Figure 6: Frontal accident with speed equal to 50 km/h

REFERENCES

Antao B.A.A., F.M. El-Turky and R.H. Leonowich. 1993. "*Mixed Simulation of Phase Locked Loops*." In Proceedings of the Custom Integrated Circuit Conference. IEEE, 8.4.1-8.4.4.

Brambilla L. 1982. "*Erhoehung der Insassensicherheit durch Airbag und Gurtstrammer*." Automobiltechnische Zeitschrift 84. 77ff.

Connelly J.A. and P. Choi. 1992. "*Macromodeling with SPICE*." Prentice Hall, Englewood Cliffs, New Jersey 07632

Eldo 1992. "*ELDO Electrical Circuit Simulator Version 4.1.x*." User's Manual. ANACAD Computer Systems (Jul.).

FAS 1992. "*ELDO-FAS Dynamical System Modeling Version 4.1.x*" User's Manual. ANACAD Computer Systems (Jul.).

Gajski D.D. , N.D. Dutt, A.C-H. Wu and S.Y-L. Lin. 1992. "*High-Level synthesis, Introduction to Chip and System Design*." Kluwer Academic Publishers.

Gielen G., E.Liu, A.L. Sangiovanni-Vincentelli and P.R.Gray. 1992. "*Analog Behavioral Models for Simulation and Synthesis of Mixed-Signal Systems*." In Proceedings of the International Conference on Computer-Aided Design. IEEE, 464-468.

Liu E., A.L. Sangiovanni-Vincentelli, G.Gielen and P.R.Gray. 1991. "*A Behavioral Representation for Nyquist Rate A/D Converters*." In Proceedings of the International Conference on Computer-Aided Design. IEEE, 386-389.

MAST 1989. "*MAST Reference manual Release 2.2*." ANALOGY Inc.

Morin J.P., F.Lémery, E.Nercessian, V.Sharma, J.Benkoski and D. Samani. 1993. "*A Practical Approach to Top/Down Analog Cicuit Design*." In Proceedings of the Nineteenth European Solid-State Circuits Conference (Sevilla-Spain, Sep.). 49-52.

Moser E. 1993. "*Specification of a first set of system components*." Milestone report. JESSI AC 3 "HDL Component Modeling Libraries" (Mar.).

Tahawy H.El., D.Rodriguez, S.Garcia-Sabiro and J.J.Mayol. 1993. "*VHDeLDO: A New Mixed Mode Simulation*." In Proceedings of the International Conference on Computer-Aided Design. IEEE, 546-551.

UNICAD 1993. "*UNICAD 2.2.D Mixed-Mode Environment User's Guide*." SGS-THOMSON Microelectronics (Dec.).

Verilog 1991. "*Verilog-XL Reference Manual*." CADENCE (Mar.).

Annexe 4: Alimentations à découpage

Publication "***Analog and Mixed Modeling with HDL-A: An Evaluation***"
soutenue à la conférence
"***Workshop on Libraries, Component Modelling and Quality Assurance***"
IRESTE - IHT, Atlanpole - La Chantrerie, Nantes, France, April 26-27, 1995
in Proceedings pp.153-166

Analog and Mixed Modeling with HDL-A: An Evaluation[1]

F. Lémery**, J.P. Morin, E. Nercessian
SGS-THOMSON Microelectronics
BP16 - 38921 Crolles cedex - France
** jointly with INPG/TIMA
46, av Félix Viallet - 38031 Grenoble cedex - France

Abstract

This paper presents modeling experiences with HDL-A, a new Hardware Description Language dedicated to analog and mixed applications. It has been designed by Anacad on the basis of the design objectives for IEEE 1076.1, called VHDL-A, but it does not yet provide all the functionality of the future standard. However, it has been successfully used to analyse the transient and frequency behavior of DC/DC converters. HDL-A mixed modeling capabilities are described on a typical regulation system, and the averaging modeling technique, used for AC analysis of switch power stages, is carried out with both a SPICE macro-model and an HDL-A behavioral one. This last approach appears much more efficient for algorithmic description convenience and simulation efficiency. Moreover, our analog and mixed design environments, fitting top/down methodology, have been extended to manage such a behavioral language.

1 Introduction

Due to telecommunication, image processing or automotive application growth, chips now integrate more analog functions, such as A/D and D/A converters or phase locked loops. In order to design such complex circuits, the top/down methodology has to be applied for both the digital and the analog parts of mixed VLSI circuits. In this methodology, which is based on a hierarchical decomposition of the system, each block is described first in a behavioral way in order to define specifications, and in a following step, in a structural manner for architecture study [1]. The standard hardware description language, VHDL, supports this methodology for the digital circuits at all the design levels.

In the analog domain, no standard is yet available. Nowadays, the most frequently used

1. This work was partly supported by the Joint European Submicron Silicon Initiative (JESSI) AC3: HDL Component Modeling Libraries

description language is certainly the input language of the SPICE-like simulators, also named SPICE and which supports only structural descriptions. Some macro-modeling techniques, named “build-up” and “simplification” have been developed in order to solve simulation problems of large circuits [2]. The “build-up” method uses the simulator primitives equations and network Kirchhoff laws to define relationships between variables that are represented by node voltages or branch currents [3]. The “simplification” method keeps the global circuit structure and replaces most of the non-linear elements by ideal ones, such as controlled sources and other passive components. Although macro-modeling is very helpful in analog simulation, it generally requires fine tuning and is not suited for complex and mixed component modeling.

Recently, behavioral modeling has been taken into account by some proprietary languages, such as FAS [4] which supports only analog behavioral descriptions, or MAST [5] which is able to manage both types of representation for mixed circuits. Circuits such as phase locked-loops [6] and A/D converters [7] have thus been modeled. Nevertheless, these languages are not as closed to the standardization requirements for VHDL-A [8], as the new language HDL-A is [9]. Its major characteristics are addressed in the first part of this paper.

In the second part, our first experiences with this language, about the modeling of some switching power supplies or DC/DC converters, are presented. A mixed functional model of the regulator block used for transient analysis is first described. Then, example of the averaging technique [10] applied to a boost converter is given, both for a SPICE macro-model and for an HDL-A behavioral model.

Finally, the new features of our top/down design environment, that proved to be required when using a mixed analog-digital HDL, are addressed.

2 HDL-A overview

It is important not to confuse the HDL-A product with the future standard VHDL-A or VHDL 1076.1 language which is still under development. Thus, no analog extensions of VHDL have yet been formally adopted as the IEEE ballot has not yet occurred. HDL-A implements some of them as they were defined at the time when it was designed. Thus, the current HDL-A version does not provide all the functionality of VHDL-A.

VHDL-A will support behavioral and structural description of mixed circuits from the system level to the component one. Digital descriptions will be fully compatible with VHDL 1076 and analog extensions will address lumped but not distributed elements belonging to

whatever physical domain (microwave circuits are under the scope of a new language MHDL). One design objective was to use, as much as possible, constructs provided by VHDL but new concepts concerning analog objects have to be introduced [11]. Some of them are presented hereafter on the HDL-A example.

The HDL-A version, we used in this work, is limited to the behavioral description of mixed circuits: analog structural description is still done using SPICE syntax. Moreover, the digital description is a small subset of VHDL and must use only one *process* with a fixed structure. However, all these limitations should be overridden in the next versions. Note that, at the moment, the Eldo simulator [12] manages both analog and digital descriptions.

Concerning the major analog extensions, HDL-A proposes a new object named *state* which is associated to a new type named *analog*. Indeed analog objects have particular properties: they can have both a temporal and a frequencial representation and thus their type can be either *real* or *complex*, they are continuous in time or frequency, their values are determined by ordinary differential equations, derivative operators can be applied on them and finally they contain an historic.

A behavioral model uses these analog quantities in order to define input-output relationships. Two types of connection points are implemented in HDL-A: *pin* and *coupling*. A *pin* is an analog interface which is characterized by two values: the value of the *across* quantity of the node to which it is connected, and the value of the *through* quantity flowing in the *pin* from the outside. Moreover, a *pin* is associated to a physical law implying energy conservation, such as that of Kirchhoff in the electrical realm. Note that a *pin* can belong to other physical domains. The *coupling* connection is implemented in order to transmit any analog quantities between several models. These two connection types are required to implement *across* and *through* controlled models.

In HDL-A, the analog relationships are described in a block, named *relation*, which is composed of *procedural* sections for the sequential algorithmic parts, and *equation* sections for the differential and implicit equations requiring iterations. These sections are associated to analysis modes, *DC, AC*, and *transient*.

3 Behavioral modeling of DC/DC converters

In order to evaluate HDL-A analog modeling capabilities, the modeling of DC/DC con-

verters has been proposed. Such circuits are composed of two parts (Fig.1): the power stage, composed of an inductor, a capacitor, a diode and a transistor, and the feedback regulator stage consisting of a pulse width modulator controlled by an error amplifier and an output current limiter. Fig.1 depicts three types of power stages: step-up or boost, step-down or buck, and fly-back or buck-boost.

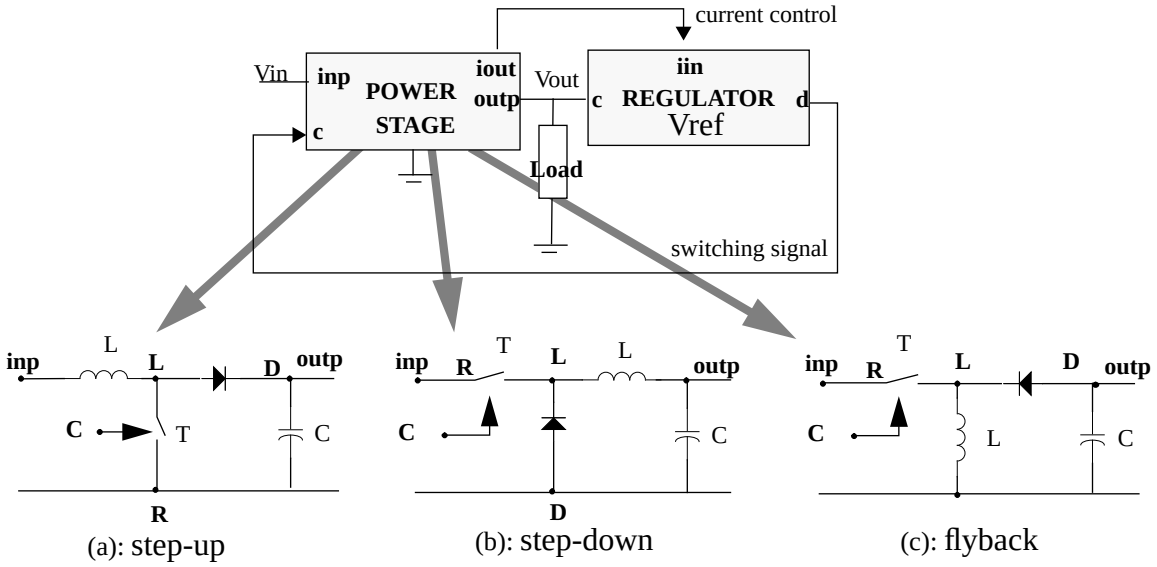


Figure 1: Basic switching converter structure, step-up, step-down & flyback

3.1 Transient behavioral modeling

Transient simulation of such a circuit is required for studying the regulation function. However, it is very time-consuming due to the high switching frequencies. In order to tackle such a problem, behavioral modeling has been carried out either with the HDL-A behavioral modeling language or with macro-modeling techniques.

Considering first the power stages, we built for each of them both an HDL-A behavioral model describing the ideal behavior of the internal components, and a macro-model composed of Eldo primitives (inductor, capacitor, ideal diode and a switch instead of the transistor). We compared their simulation times and we noticed that the macro-model was much more efficient: the power stages are indeed very low-level components and, in this case, macro-modeling with simulator built-in primitives, is more powerful.

On the other hand, HDL-A allowed us to model the mixed regulator, depicted at Fig.2, at a functional level with high efficiency.

A special procedure, named soft start function, is used in some SGS-Thomson step-down

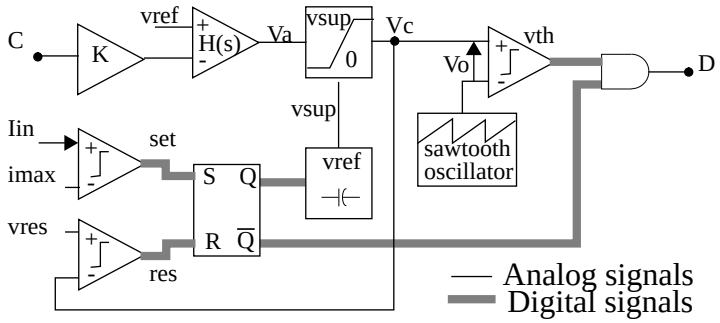


Figure 2: Simplified regulator structure of a step-down converter

regulators [13] in order to prevent output overcurrents at switch on. In the actual circuits, the load current is sensed by an internal resistor connected to a comparator, while, in our case, it is measured through a null voltage source set in the macro-model of the power stage. When the load current exceeds a threshold (*imax*), the RS-flip-flop is set, which disables the output D and causes the clamping voltage (*vsup*) of the error amplifier output to decrease. When this voltage has fallen to a small threshold (*vres*), the flip-flop is reset: the output is thus enabled and the clamping voltage rises up to *vref*. The main analog and digital waveforms are presented in Fig.3. The HDL-A code which is based on Fig.2, is as follows:

```
ENTITY regulator IS
    GENERIC(k, ad, n1, d1, ccste, dcste, vref, imax, vhigh, vlow: real;
            per, td: time);
    COUPLING( iin: analog);
    PIN ( c, d: electrical);
END ENTITY regulator;

ARCHITECTURE TR OF regulator IS
 CONSTANT vth, vmax, vres, ks: REAL;
 VARIABLE start, tau: REAL;
 SIGNAL s_v, set, res, clk, q, s_on : BIT;
 STATE ve, va, vsup, vlim, vc, vsaw, vo: ANALOG;
 BEGIN
   PROCESS
   BEGIN
      -- initialization at the beginning of the simulation
      clk <= '0', '1' after per;
      IF vo>vth THEN s_v<='1'; ELSE s_v<='0'; END IF;
      set <= '0'; res <= '0'; q <= '0'; s_on <= s_v;
   LOOP -- infinite loop
    WAIT ON clk, s_v, set, res, q, RISING(vo,vth),FALLING(vo,vth),
      RISING(iin,imax),FALLING(iin,imax),RISING(vc,vres),FALLING(vc,vres) ;
      -- digital oscillator signal
      IF EVENT(clk) THEN clk <= NOT clk AFTER per; END IF;
      -- pwm voltage comparator
      IF RISING(vo,vth) THEN s_v <= '1' AFTER td;
      ELSIF FALLING(vo,vth) THEN s_v <= '0' AFTER td;
      END IF;
      -- current comparator
      IF RISING(curr,imax) THEN set <= '1' AFTER td;
      ELSIF FALLING(curr,imax) THEN set <= '0' AFTER td;
      END IF;
      -- reset comparator
      IF FALLING(vc,vres) THEN res <= '1' AFTER td;
      ELSIF RISING(vc,vres) THEN res <= '0' AFTER td;
      END IF;
      -- rs flip-flop
```

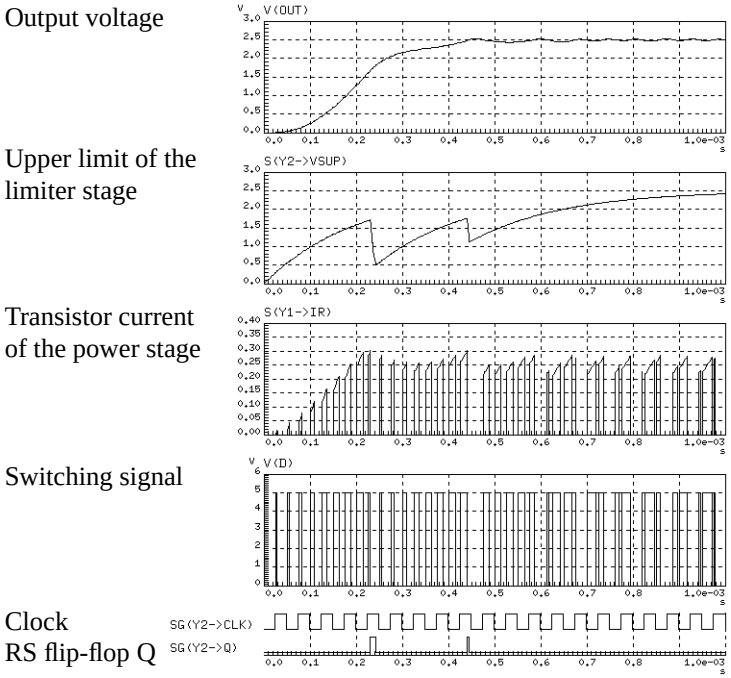
```
      IF set='0' AND res='1' THEN q <= '0'AFTER td;
      ELSIF set='1' AND res='0' THEN q <= '1'AFTER td;
      END IF;
      -- and ouput gate
      s_on <= s_v AND NOT q AFTER td;
    END LOOP;
    END PROCESS;

  RELATION
  PROCEDURAL FOR INIT =>
    ...
  PROCEDURAL FOR DC =>
    ve := vref - k*c.v; d.v %= 0.0;
  PROCEDURAL FOR TRANSIENT =>
    ve := vref - k*[c,gnd].v;
    -- dynamic limiter
    IF q='0' THEN vlim := vref; tau := ccste; -- charge: vsup rises
    ELSE vlim := 0.0; tau := dcste; -- discharge: vsup: falls
    END IF;
    IF va>vsup THEN vc:=vsup; ELSE vc:=va; END IF;
    -- sawtooth wave generation
    IF EVENT(clk) THEN start := CURRENT_TIME; END IF; -- new rising edge
    vsaw := ks*ANALOG(CURRENT_TIME-start); vo := vc - vsaw;
    -- output stage
    IF EVENT(s_on) THEN -- ideal D/A conversion
      IF s_on='1' THEN d.v %= vhigh; ELSE d.v %= vlow; END IF;
    END IF;
  EQUATION (va,vsup) FOR DC, TRANSIENT =>
    -- amplifier transient response H(s) = va/ve = ad(1+n1.s)/(1+d1.s)
    d1*DDT(va)+va == ad*(n1*DDT(ve)+ve);
    -- vsup is modeled as the charge/discharge of a capacitor
    tau*DDT(vsup)+vsup == vlim;
  END RELATION;
END ARCHITECTURE tr;
```

This regulator model is described in a signal flow manner: the currents flowing through the input pin *c* and output *d* are not considered. The duty-cycle of the switching signal is controlled by the voltage of the pin *c*, denoted in HDL-A *c.v*, and also by the current flowing in a branch of the power stage, which is transmitted to the regulator through the *coupling* connection *iin*. The output analog signal is applied on *d* using the new assignment *%=* which implies a contribution (of voltage or current) on a pin.

The analog *relation* part describes the behavior of an input error amplifier, whose transfer function is modeled by a differential equation, a dynamic limiter, whose upper limit (*vsup*) depends on the state of the RS-flip-flop and which is modeled by another differential equation, and finally the sawtooth oscillator, whose rising edge is a function of the analog simulator time denoted here *current_time*.

The single digital *process* generates the clock and models ideal comparators, an RS-flip-flop and the output AND gate. The *process* has a fixed structure with a *wait* statement, whose position determines the separation of the initialization block from the simulation one, based on an infinite *loop* statement. Thus, the initialization part is only carried out at the beginning of the simulation, while the second part is active during all the simulation.

Concerning the analog-digital interactions, new functions (*rising* and *falling*) are provided in order to detect analog events, such as threshold crossing, in the digital *process*. Digital events on signals are also detected with the *event* function. Digital signals are only assigned in the *process* but can be read in the analog equation block. Similarly, analog objects are only assigned in the analog part, but their values can be used in the digital *process*.

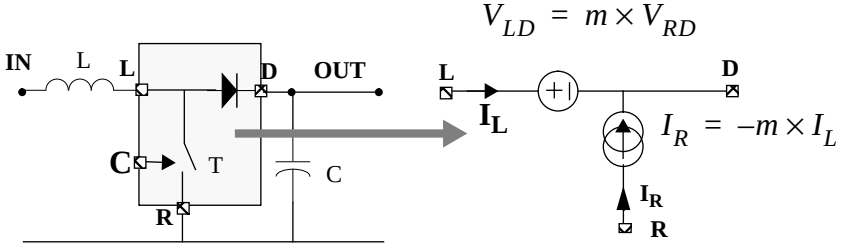


Vin=5V, Vref=2.5V, Imax=300mA, T=25ms, Rout=10W, C=10mF, L=1mH

Figure 3: Example of a *step-down* converter regulation

3.2 AC behavioral modeling

System designers also need to analyse the stability of the power converters in the frequency domain [14]. An averaging model of the power stage (Fig.4) which computes the averaged voltages and currents has to be used.



For continuous conduction: $\frac{Vout}{Vin} = \frac{Iin}{Iout} = \frac{1}{1-m}$

Figure 4: Power stage of a boost converter and its averaging model

This model is based on an ideal transformer composed of a voltage controlled voltage source V_{LD}, instead of the diode, and a current controlled current source I_R, instead of the

switch transistor. The transformer ratio, *m*, represents the duty-cycle, when the converter is working in continuous conduction mode.

Moreover, the model has to determine the conduction mode of the converter. Indeed when the load becomes too high, the inductor current can become zero and the converter goes from the continuous to the discontinuous conduction mode. For the continuous mode, the absolute value of the averaged transistor current, I_R, is equal to $\left|I_L \times V_C\right|$ and for the discontinuous one, it is equal to $\left|V_{LR} \times V_C^2 \times \frac{T}{2 \times L}\right|$ where T is the switching signal period and L the inductor. The conduction mode is just the one for which I_R is the greatest.

3.2.1 Classical macro-model approach

Fig.5 presents the principle of a SPICE macro-model as described in [15], with the "built-up" method.

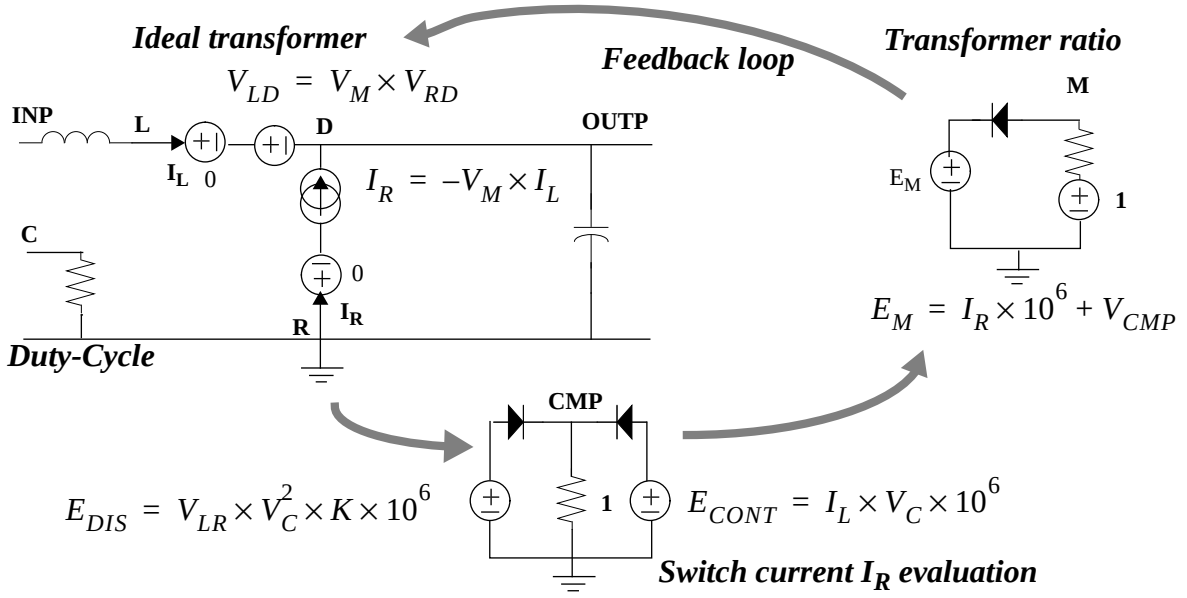


Figure 5: SPICE macro-modeling of an averaging boost power stage

The diode scheme performs the I_R current comparator and a feedback loop is implementedin order to compute the unknown transformer ratio, *m*, represented by the voltage on the node M. Currents are sensed through zero voltage sources, and mathematical operations are performed using controlled sources (in the initial SPICE simulator, only polynomial expressions can be used). In order to avoid the diode threshold effect, some variables must be scaled by an 1e6 factor.

Macro-modeling is thus based on basic electronic laws and enables one to perform quite complex mathematical operations. However, this non-linear and highly coupled circuit can

rapidly induce some convergence troubles in the simulator.

3.2.2 HDL-A approach

Such an algorithmic model is in fact more conveniently described with an analog behavioral modeling language such as HDL-A. We shall now present the example of the boost power stage:

```
ENTITY power_stage IS
  -- entity of an average model of switch DC/DC converter
  generic ( ind, cap: real; per: time); -- parameters
  pin ( inp, outp, c, gnd : electrical); -- analog connection points
END ENTITY power_stage;

ARCHITECTURE boost OF power_stage IS -- behavioral description
   -- variable declarations
  constant k : real;
  variable dmode : boolean;
  state vind, icap, vld, vrl, vrd, vc : analog; -- analog signals
  state ir, ircont, irdis, il, m : analog;
  BEGIN
  RELATION
      PROCEDURAL FOR INIT => -- parameter default values definition
         ind :=1.0e-3;
         cap :=1.0e-6;
         per :=100us;
         k := real(per)/(2.0*ind); -- constant definition
      PROCEDURAL FOR DC, AC =>
       -- read voltages on the connection pins:
         icap := cap*ddt([outp,gnd].v); -- capacitor current
         vind := ind * ddt(il); -- inductor voltage ( il is an unknown)
         vld := [inp,outp].v - vind;
         vrl := [gnd,inp].v + vind;
         vrd := [gnd,outp].v;
         vc := [c,gnd].v;
         irdis := k*vrl*vc*vc; -- discontinuous current value
         ircont := -vc*il;  -- continuous current value
      PROCEDURAL FOR DC => -- conduction mode determination in DC analysis:
         if (abs(irdis) > abs(ircont)) then dmode:=true;
            else dmode := false;
         end if;
      PROCEDURAL FOR DC, AC => -- switch current ir evaluation:
         if dmode then ir := irdis;
         else ir := ircont;
         end if;
         -- apply currents on the connection pins:
         gnd.i %= ir-icap;
         inp.i %= il;
         outp.i %= -il-ir+icap;
      EQUATION (m, il) FOR DC, AC =>
         -- implicit equation of the ideal transformer with m & il as unknowns:
         vld == m*vrd;
         ir  == -m*il;
    END RELATION;
END ARCHITECTURE boost;
```

The power stage entity can be common to all types of switching converters (boost, buck, buck-boost) while the architecture describes each behavior type.

The architecture structure is quite general: voltages are read on the pins (*inp.v, outp.v,*

gnd.v) in order to determine the *pin* currents (*inp.i, outp.i, gnd.i*). Note that particular care must be taken on the current convention for the current Kirchhoff laws application.

Moreover, the computation of the transformer ratio *m* and of the current *il* flowing through the inductor, requires some iterations on the equation system composed of the ideal transformer relationships, described in the *equation* section, and also of the circuit equations.

3.2.3 Simulation results

These average power stage models, the macro-model and the behavioral model, can be used to compute the duty-cycle which is required to maintain a constant output voltage, whatever the load current is. In this analysis, the regulator of the feedback loop is just modeled as a high gain error amplifier with a particular reference voltage. For the macro-model, we found that the value of the feedback gain giving a correct accuracy was around 1e5. Over this value, simulation gives erroneous results. There is no such problem with the HDL-A model which can therefore give a better accuracy.

Fig.6 presents the averaged duty-cycle computed by the regulator, as a function of the load current. As expected, the duty-cycle must be reduced if the output current falls below the minimum current [16].

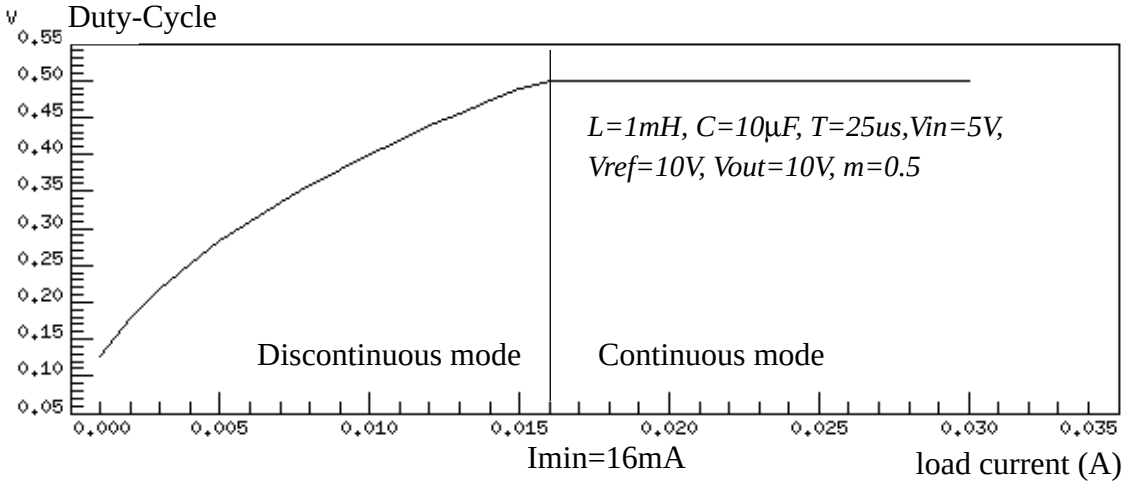
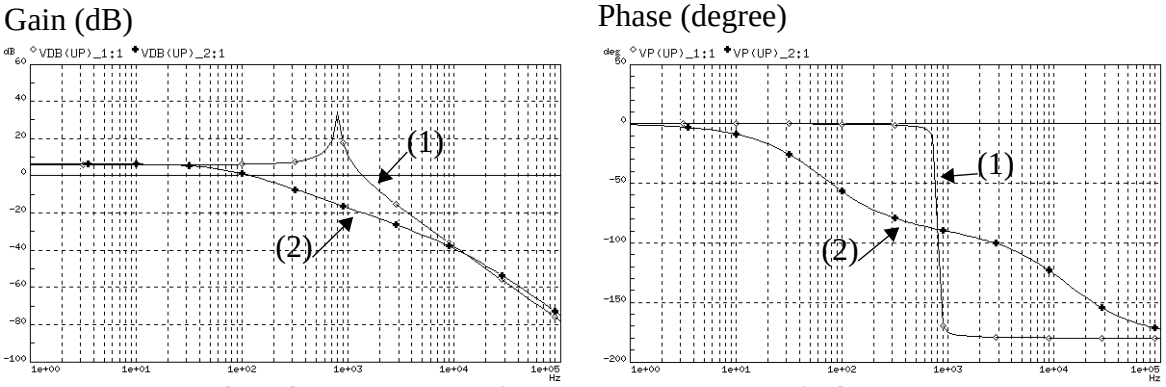


Figure 6: Duty-Cycle vs load current for boost, buck & buck-boost converters.

On a SPARCstation 2, the DC simulation takes 9s of CPU time with the macro-model, against less than 1s with the HDL-A model. In this case, behavioral modeling is more efficient than macro-modeling. Indeed, the macro-model is very complex and requires much more simulation iterations.

AC simulations have also been performed on both power stage models. Fig.7 depicts the gain and phase variations of the output voltage to small variations of the input voltage, in the two conduction modes, continuous with a load resistance of 500Ω, and discontinuous with a load of 700Ω. The duty-cycle was set to 0.5 and component values are the same as in Fig.6.



Load conditions (1) R=500Ω: continuous, (2) R=700Ω: discontinuous

Figure 7: AC response of a boost power stage for two load

Results are consistent with the theoretical study about DC/DC converter modeling [17]. In the continuous conduction mode, the second order transfer function has two complex poles, and the resonance frequency is given by $\frac{1}{2\pi}\frac{(1-\alpha)}{\sqrt{LC}}$, namely 796 Hz. The magnitude of the DC gain is of 2. In the discontinuous mode, the transfer function has two real poles, and the dominant one is given by $\frac{1}{2\pi RC}\left(\frac{2Vs}{Ve}-1\right)/\left(\frac{Vs}{Ve}-1\right)$, namely 67 Hz. Moreover, in this case the DC gain is a little bit greater than 2 (2.06).

On a SPARCstation 2, the previous AC simulation takes 50s of CPU time with the macro-model, against 4s with the HDL-A model. It is thus much more interesting to use behavioral modeling for both description convenience and simulation efficiency.

4 Analog and Mixed design environment

The SGS-Thomson analog and mixed design environments, that were already presented in [18, 19], have been improved in order to allow the user to perform behavioral modeling with HDL-A. These environments are based on Cadence Design Framework II, the analog Eldo simulator and the mixed engine Verilog-Eldo. They are also designed in order to fit the top/ down design methodology and provide an easy creation and selection of analog behavioral descriptions for circuit blocks. This lets the designer switch between structural and behavioral representations. It is now possible to simulate a circuit described with transistors, SPICE macro-models, Eldo-FAS, HDL-A, and Verilog models.

For example, when creating an HDL-A model from a graphic representation, a template is provided assuring a consistency of block parameters, pins, ports and couplings. In particularly, this template describes the entity of the model, and the initialization of the parameters within an architecture. Different architectures can also be described for one block and independently selected. After completing the HDL-A description, the model is compiled and stored in a work library specified by the user. Different libraries can be referenced for one simulation.

The designer can also display analog *states* and digital *signals* of HDL-A models, particularly for debugging purposes. The desired objects have just to be selected from a list which is automatically built by the system, for each HDL-A model configuration.

Furthermore, special attention was paid in order to set automatically converters on mixed nodes. Indeed, HDL-A digital *ports* have to be connected to analog components only through A/D or D/A converter models. In a top/down process, when switching a mixed HDL-A behavioral block to its analog structural representation, some converters are required, as depicted in Fig.8. Note that our environment does not look into the hierarchy and assumes that all structural sub-circuits are analog, even if they are composed of other mixed behavioral models. Converters are also required when HDL-A *ports* are connected to digital *Verilog* blocks. Furthermore, as HDL-A models belong to the analog world simulated by Eldo, other types of converters have to be set at the interfaces with the digital simulator.

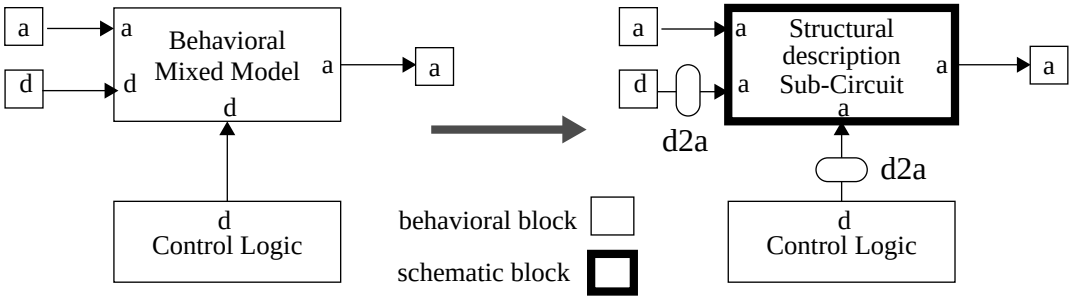


Figure 8: Switch from an HDL-A to a structural description

The converters direction is deduced from the direction of the HDL-A digital ports D/A for an output port and A/D for an input one. A converter model is also defined by its name and its mode, either *bit*, *slope* or *real*. At the moment, only the digital *bit* and *real* types are available. Some default converter models are defined in a dedicated file but the list can be extended by the user, who may want to set any particular type of converter on any HDL-A digital port.

5 Conclusion

After a brief overview of the new mixed behavioral modeling language HDL-A, we have studied its capabilities on the modeling of some DC/DC converters. In particular, HDL-A let us describe easily, at a functional level, the time behavior of an SGS-Thomson analog/digital regulator. HDL-A let us also build averaging models of the switch power stages, in order to perform frequencial analysis. Thus, this new language, which can be considered as a prototype of the future IEEE standard VHDL-A, was found to be very convenient for these applications.

Moreover, we have considered its impact on SGS-Thomson design environment, dedicated to high-level mixed modeling and simulation. New features have been implemented in order to tackle the analog-digital conversion problem, which occurs when applying the top/down design methodology on mixed circuits.

However, such problems should be solved with an unified language for analog and digital, behavioral and structural descriptions, such as VHDL-A and the next HDL-A versions.

Acknowledgment

The authors wish to thank O.Schnitzler for HDL-A support, R.Stewart and J.Scully for manuscript review.

References

[1] D.D.Gajski, N.D.Dutt,A.C-H. Wu, S.Y-L. Lin, *High-Level Synthesis, Introduction to Chip and System Design*, Chapter 1, Kluwer Academic Publishers, 1992

[2] G.R.Boyle, B.M.Cohn, D.O.Pederson, J.E.Solomon, *Macromodeling of Integrated Circuit Operational Amplifiers*, IEEE Journal of Solid-State Circuits, Vol. 9, No. 6, December 1974

[3] J. A. Connelly, P. Choi, *Macromodeling with SPICE*, Prentice Hall 1992

[4] MAST reference manual Release 2.2 ANALOGY Inc., April 1989

[5] ANACAD, ELDO-FAS Dynamical System Modeling, Version 4.1.x, July 1992

[6] Antao B.A.A., F.M. El-Turky and R.H. Leonowich. 1993. *Mixed Simulation of Phase Locked Loops*. In Proceedings of IEEE 1993 Custom Integrated Circuit Conference, p.8.4.1-8.4.4.

[7] G. Ruan, *A behavioral model of A/D Converters using a mixed-mode simulator,* In Proceedings of IEEE 1990 Custom Integrated Circuits Conference, p.5.7.1-5.7.4

[8] IEEE VHDL subPAR 1076.1 Analog Extensions to VHDL Design Objective Document (DOD), Version 2.1, Nov. 8, 1994

[9] ANACAD, HDL-A Language Reference Manual, July 1994

[10] Voperian, *Simplified Analysis of PWM converters using the model of the PWM switch, Part 1: continuous conduction mode,* IEEE Transactions on AES, Vol 26, No.2, March 1990

[11] Resve A. Saleh & al., *Analog Hardware Description Languages,* In Proceedings of IEEE 1994 Custom Integrated Circuits Conferences, p.15.1.1-15.1.8

[12] ANACAD, ELDO User's Manual, Advanced Analog and Mixed Signal Solutions, Issue 4.3, June 1994

[13] SGS-THOMSON Microelectronics, *Industry Standard Analog ICs Databook*, May 1993, p.173-177

[14] K.J. Karimi, Boing Computer Services, *Modeling and simulation of large DC power electronics systems*, In Proceedings of the Conference on Modelling and Simulation 1994, p.1111-1115

[15] D.J. Caldwell, *Techniques let you write general-purpose Spice models*, EDN September 1991, p.149-154

[16] U.Tietze, Ch.Schenk, *Electronics Circuits Design and Applications*, Springer-Verlag, Berlin, Heidelberg 1991, p.505

[17] J.P. Ferrieux, *Modélisation des convertisseurs continu-continu à découpage*, Thèse de l'INPG, 4 mai 1984

[18] J.P. Morin, F.Lémery, E.Nercessian, V.Sharma, J.Benkoski and D.Samani, *A practical approach to Top/ Down analog circuit design*, In Proceedings of the Nineteenth European Solid-State Circuits Conference 1993, p.49-52

[19] F.Lémery, J.P.Morin, E.Nercessian, J.Benkoski, D.Samani, *Behavioral models for complex top/down analog/digital system simulation*, In Proceedings of the Conference on Modelling and Simulation 1994, 1050-1054

Annexe 5: Environnement de modélisation

Publication "***An Interactive Environment for Analog Characterization and Behavioral Modelling***"

présentée sous forme de poster à la conférence

"***ESSCIRC'95 Euroconference, 21st European Solid State Circuits Conference***"

Lille, France, 29-22 September 1995

in Proceedings *ESSCIRC'95, pp.314 - 317*

An Interactive Environment for Analog Characterization and Behavioral Modeling[1]

F. Lémery**, J.-P. Morin, E. Nercessian

SGS-Thomson Microelectronics
BP 16 - 38921 Crolles cedex - France
** jointly with INPG/TIMA
46, av Félix Viallet - 38031 Grenoble cedex - France

ABSTRACT In order to help analog designers to apply the top-down methodology, a new interactive environment is proposed. It provides a functional library for both high-level and low-level modeling. Model parameters can be computed from circuit characterization data, the characterization program being automatically generated from parsing parameter expressions. Characterization procedures are saved in a library and can also be used only for data-sheet extraction. Finally, language translators are provided to migrate behavioral models from one language to another.

1 Introduction

Design of today's analog and mixed ICs requires to use the top-down methodology which is based on behavioral representation of circuit blocks [20]. It allows to deal with complex circuits and speed-up simulation. Some algorithms have already been presented in the past, concerning automatic equation generation [21] and optimization [22] of macro-models, but they are limited to known structures. Characterization tools have also been described [23][24] but the relationship with macro-model parameters is not obvious.

Therefore, we developed an interactive and practical environment, which intends to help designers to develop easily behavioral models and to compute automatically their parameters from circuit characterization data. We already used this environment for both the description of a complex air-bag system and the macro-modeling of an SGS-Thomson op-amp.

2 Overview of the environment

The environment for behavioral modeling development is depicted in Figure 1. It is based on Cadence Design Framework II. Inputs are a circuit netlist, which is considered as a black-box, and either a macro-model schematic (input A) or an *analysis* list (input B).

A macro-model schematic can be built using blocks from a ***functional library*** in a block diagram manner [25]. Each block is described either as a Spice macro-model or as a behavioral Fas [26] model. Model equations are automatically generated by a ***"netlister"*** which checks the language property of each block. Finally, Fas behavioral models can be translated into HDL-A [27], which satisfies most of the design objectives of VHDL-A [28], or CFas [29], which improves simulation speed, or Mast [30].

Macro-model parameters can be defined by the user, as mathematical functions of circuit mea-

1.This work was partly supported by the Joint European Submicron Silicon Initiative (JESSI) AC12

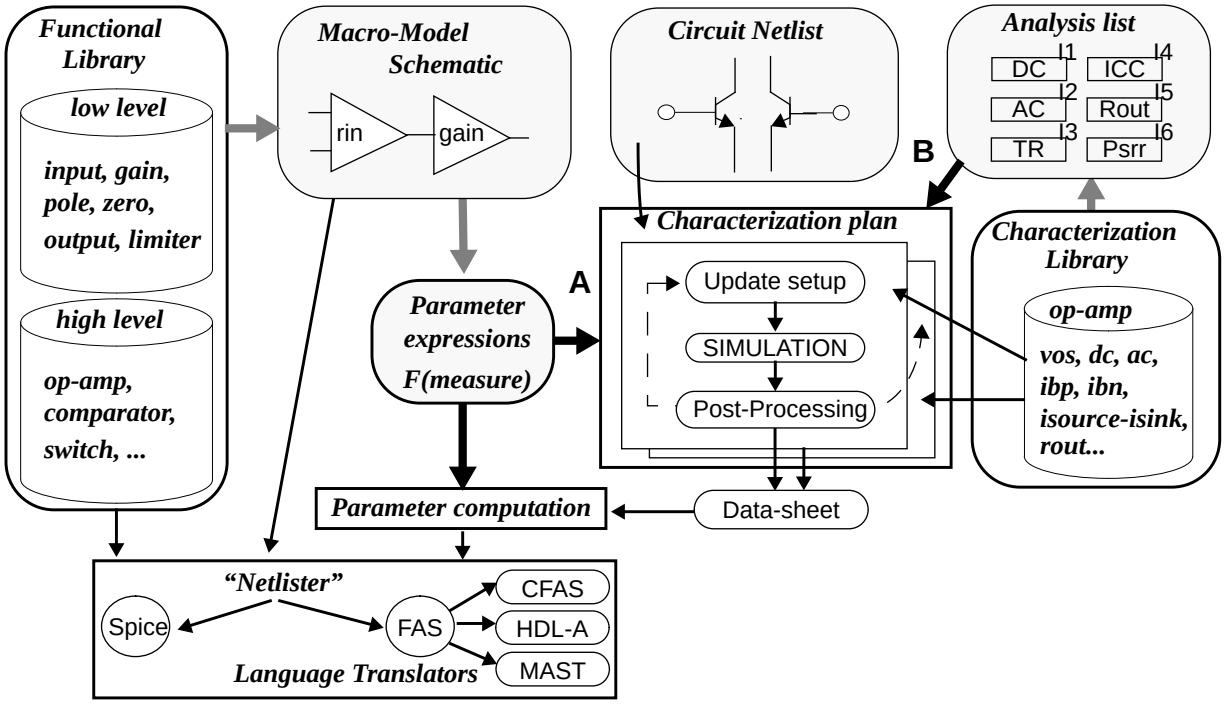


Figure 1 Behavioral-modeling environment

surements, denoted ***F(measure)*** in Figure 1. These expressions are parsed in order to generate automatically both a ***parameter computation*** program and a circuit ***characterization plan***. This plan takes into account characterization procedure inter-dependencies. For instance, open-loop *AC* characterization of an op-amp needs the offset value for compensation, the offset being measured with an external *DC analysis*.

Characterization procedures are stored in a ***characterization library*** and can be instantiated in a "schematic" (input B), in order to perform automatically circuit data-sheet extraction.

3 Behavioral model equation generation

3.1 The functional library

This library of functional parameterized technology-independent models is composed of both high-level models, such as mathematical functions, op-amps, converters [20], and lower-level blocks, such as basic internal circuit stages (input, gain, pole, zero, voltage saturation, output) [25]. These lower-level components can be used to build a complex macro-model schematic. The objective is to provide a model for each phenomena type occurring in transistor-level circuits. For instance, the *input* block models current and voltage offsets, bias currents and input impedance; the *slew-rate* block models a pole and voltage derivative limitation.

3.2 Language translators

An environment for automatic translation building had already been developed at SGS-Thomson for netlist languages [31]. However, behavioral analog HDLs are not semantically equivalent, as netlist ones are. Special data handling have therefore to be performed.

The main translation problems concern model structure, especially variable declaration, definition of analysis mode dependent sections (DC, AC, transient), and usage of equations requiring simulator iterations. Information can be translated by sorting, and adding new intermediate statements and variables. Variable type translation is another difficulty. As the input language is read in one pass and back-tracking is difficult to carry out, simple rules have been defined, taking into account the most

common cases. But manual modifications can be required for typical applications. Finally, for some languages (Mast), dedicated simulator information have to be added manually.

4 Model parameter computation

An analog cell characterization module is used to extract circuit data-sheet performing a set of several characterization procedures, that are stored in a library.

4.1 Characterization library

It allows designers to save their characterization knowledge. It is composed of measurement procedures, or *analyses*, that are carried-out by the characterization program. Each procedure extracts a set of circuit characteristics with a particular *analysis* parameterized scheme. Extractions are performed using functions of Eldo simulator [32]. *Analysis* setups involve component values of the *analysis* schematic, stimuli and simulator commands. Note that a setup value can depend on an external measurement. An iterative procedure can be defined in order to find the setup range which allows measurement.

At the moment, a library has been developed for op-amp circuit class, but all facilities are provided to build new characterization procedures through a friendly user interface.

4.2 Parameter expressions

A dedicated syntax has to be used to tell the system that a behavioral model parameter depends on characterization data. It allows the user to describe together the name of the characteristic which has to be measured, the name of the analysis to be carried-out, optionally its setup configuration name, and finally the default value of the measure.

For instance, one could define the differential gain of an amplifier macro-model as the average value of the differential gain characteristics, extracted by an *AC analysis*, in two different setup configurations. This mathematical expression will be parsed to produce the corresponding characterization commands, involving two *AC analyses*, but also two offset voltage measurements, that are pre-requisite of *AC* analyses. Consistent setup configurations will have to be defined by the user before launching the characterization program.

5 Evaluation example

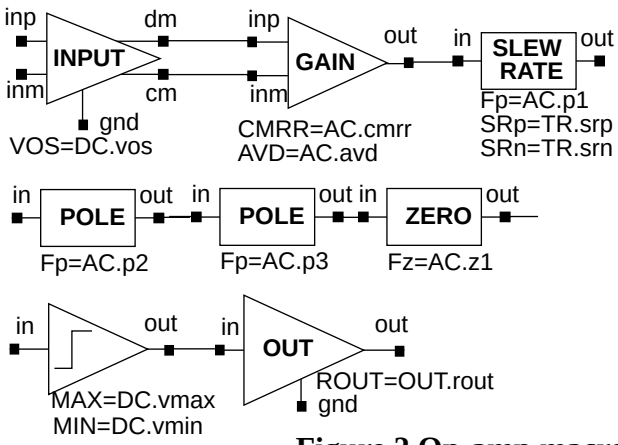
This environment has been used for the modeling of an electronic triggering device for an air-bag system [33]. This device is in particular composed of an analog-digital block, named *data-path*, whose architecture has been described using high-level blocks of the functional library (high-pass filter, limiting amplifier, substractor, comparator), and some other dedicated behavioral models (bounded integrator and output stage), written first in Fas and automatically translated in HDL-A.

We also described architecture of some components. For instance, the bounded integrator is based on an op-amp, whose macro-model has been built with the block diagram approach, as described in Figure 2. Apart of the input and output stages, that apply currents respectively on input and output pins, all other blocks are described in a signal flow manner. In particular, the transfer function is described with gain, slew-rate, pole, and zero stages.

Macro-model parameters have been computed through characterization of an SGS-Thomson fast op-amp. The macro-model has been also characterized with the same procedures. Table of Figure 2 shows that this block diagram modeling approach, which is very flexible, provides a good accuracy. Moreover, simulation speed is significantly improved.

6 Conclusion and future work

In this paper, we have presented a new environment for behavioral modeling, which is based on a



Data	Circuit	Macro
Maximum V.	14.8 V	14.8 V
Minimum V.	-14.2 V	-14.2 V
Pos. slew-rate	15.4 V/us	13.6 V/us
Neg. slew-rate	-11.9 V/us	-10.5 V/us
Diff. Gain	102.8 dB	102.8 dB
CMRR	119.8 dB	119.8 dB
GBw at 0 dB	4.95 MegHz	4.96 MegHz
Gain margin	-11.22 dB	-24.5 dB
Phase margin	43.1 degree	54 degree
Rout	1766 Ohm	1766 Ohm

Figure 2 Op-amp macro-model example

large functional library, composed both of high-level and low-level blocks, behavioral language translators, an analog cell characterization tool, and a characterization library.

This environment extends the current capabilities of SGS-Thomson analog design environment and will relieve designers from tedious and time-consuming tasks. Extension of both the functional model library and characterization one is going on, in order to cover more applications. We intend also to use the powerful simulation and optimization Anacad's program SimPilot [34].

References

[20] J.P. Morin, F.Lémery, E.Nercessian, V.Sharma, J.Benkoski and D.Samani, A practical approach to Top/Down analog circuit design, In Proc. of the Nineteenth European Solid-State Circuits Conference 1993, p.49-52

[21] H.A.Mantooth, P.E.Allen, A Higher Level Modeling Procedure for Analog Integrated Circuits, Analog Integrated Circuits and Signal Processing 3, 181-195, Kluwer Academic Publishers, 1993

[22] G.Casinovi, A.Sangiovanni-Vincentelli, A Macromodeling Algorithm for Analog Circuits, IEEE Transactions on Computer-Aided Design, Vol.10, No.2, February 1991

[23] S.A.Huss, M.Gerbershagen, G.Tränkle, Automatic Performance Characterization of Analog Functional Blocks, Analog Integrated Circuits and Signal Processing 1, 277-286, Kluwer Academic Publishers, 1991

[24] R.K.Henderson, M.Hinners, P.Nussbaum, L.Astier, Capture and Re-use of Analog Simulation Knowledge, In Proc. IEEE Custom Integrated Circuits Conference, pp. 15.2.1-15.2.4, 1994

[25] J.A.Connelly, P.Choi, Macromodeling with SPICE, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1992

[26] ELDO-FAS Dynamical System Modeling, ANACAD-EES, Version 4.1.x, Jul. 1992

[27] HDL-A Language Reference Manual, ANACAD-EES, Jul. 1994

[28] IEEE VHDL subPAR 1076.1 Analog Extensions to VHDL Design Objective Document (DOD), Version 2.1, Nov. 8, 1994

[29] CFAS C-language FAS interface Manual, ANACAD-EES Internal Documentation, Aug. 1993

[30] MAST reference manual Release 3.0 ANALOGY Inc., Oct. 1990

[31] P.Saramito, G.Fourneris, ZEBU: A multi-target Translator Environment, SGS-Thomson Microelectronics Central R&D, Internal Documentation, Sep. 1993

[32] ELDO User's Manual, Advanced Analog and Mixed Signal Solutions, ANACAD, Issue 4.3, June 1994

[33] F.Lémery, J.P.Morin, E.Nercessian, J.Benkoski, D.Samani, Behavioral models for complex top/down analog/digital system simulation, Proc. Conference on Modelling and Simulation 1994, 1050-1054

[34] SIMPILOT, User's Manual Revision 2.0, ANACAD-EES, Oct. 1994

Annexe 6: Schémas de mesure des Op-Amps

1 Tensions et courants parasites (VOS_test)

Les fonctions d'extraction de cette procédure, dont le schéma est décrit en Figure 1, et qui effectue Analyse du point de fonctionnement (`.op`), déterminent:

- la tension d'offset: v(s) - v1
- le courant d'entrée de la borne positive ibp=-i(v1),
- le courant d'entrée de la borne négative ibn=i(v2)

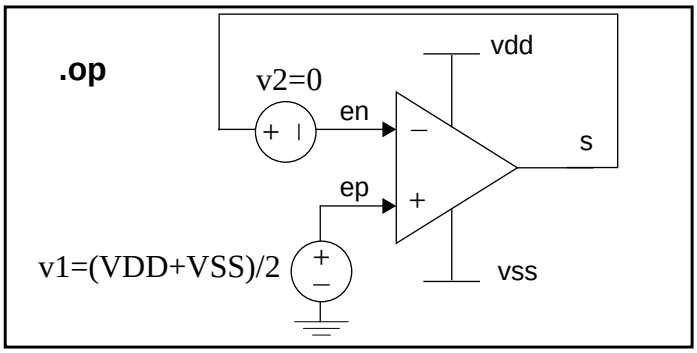


Figure 1 Schéma de la procédure VOS_test

2 Analyse DC (DC_test)

Cette procédure (cf Figure 2) permet de mesurer:

- les tensions de sortie maxi. et mini.,
- la dynamique d'entrée ou *swing*,
- le gain DC égal à la pente de *vout* lorsque v(out)=(vmin+vmax)/2: cette mesure est assez peu précise car elle dépend du pas de tension choisi pour la commande de l'analyse (`DCSTEP`),
- la tension d'offset: là encore, la précision dépend du paramètre `DCSTEP`.

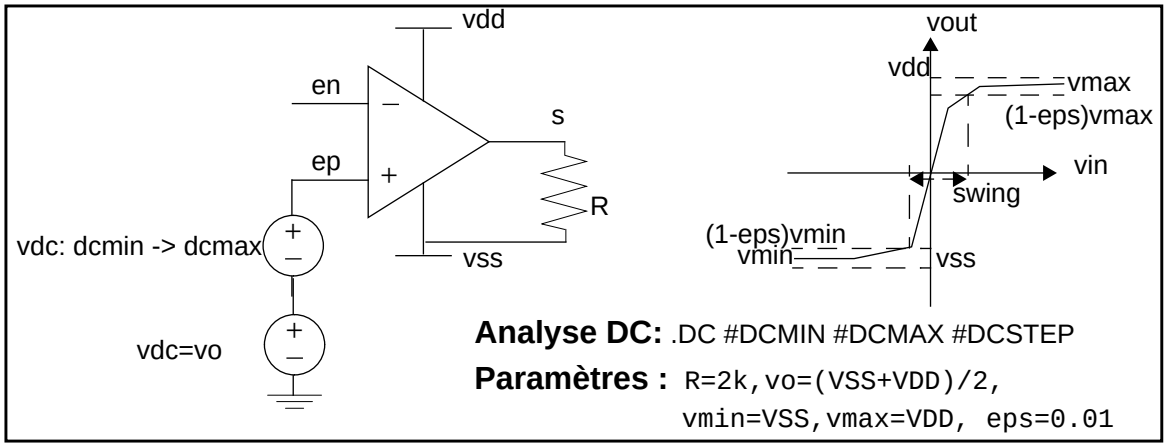


Figure 2 Schéma de la procédure DC_test

3 Analyse transitoire (TR_test)

Les paramètres temporels de l'impulsion (`tcom`, `pw`, `per`) sont calculés à partir d'une unité de temps (`tstep`). Le schéma est représenté en Figure 3. Une boucle est réalisée sur `tstep` pour trouver la plage correcte d'extraction. Les mesures effectuées sont les suivantes:

- slew-rate positif, évalué pour un front d'entrée ascendant,
- slew-rate négatif, évalué pour un front descendant,
- overshoot positif,
- overshoot négatif,
- temps d'établissement positif (temps à partir duquel v(out) entre dans la plage [vmax.(1-eps), vmax.(1+eps)] où eps est un paramètre de précision,
- temps d'établissement négatif (temps à partir duquel v(out) entre dans la plage [vmin.(1+eps), vmin.(1-eps)].

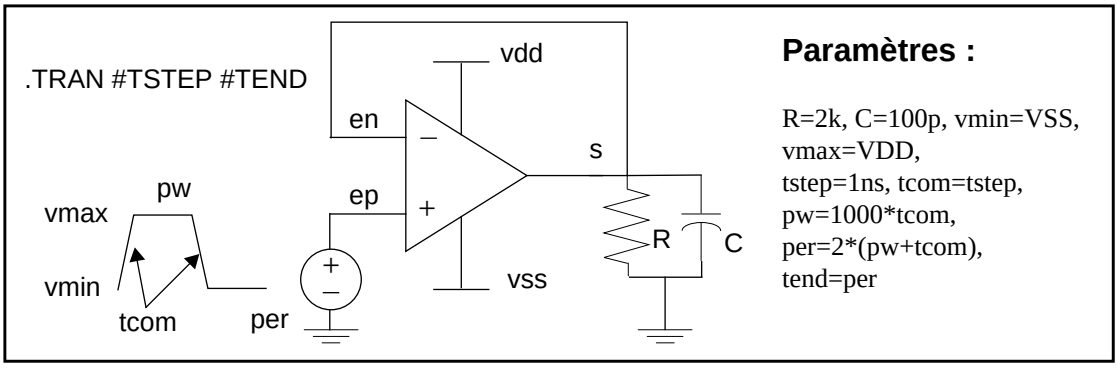


Figure 3 Schéma de la procédure TR_test

4 Analyse AC (AC_test)

La même procédure permet de caractériser les fonctions de transfert en modes communs et différentiels (cf Figure 4). En particulier, *AC_test* retourne:

- le gain différentiel DC en dB,
- le facteur de réjection de mode commun ou CMRR en dB,
- le produit gain-bande ou *unity gain bandwidth*: ugb
- la fréquence de marge de gain (en Hz): fmg
- la marge de gain (en dB): mg
- la marge de phase (en degré): mph
- la liste des poles-zéros de la sortie différentielle, obtenue par une analyse spécifique après la simulation.

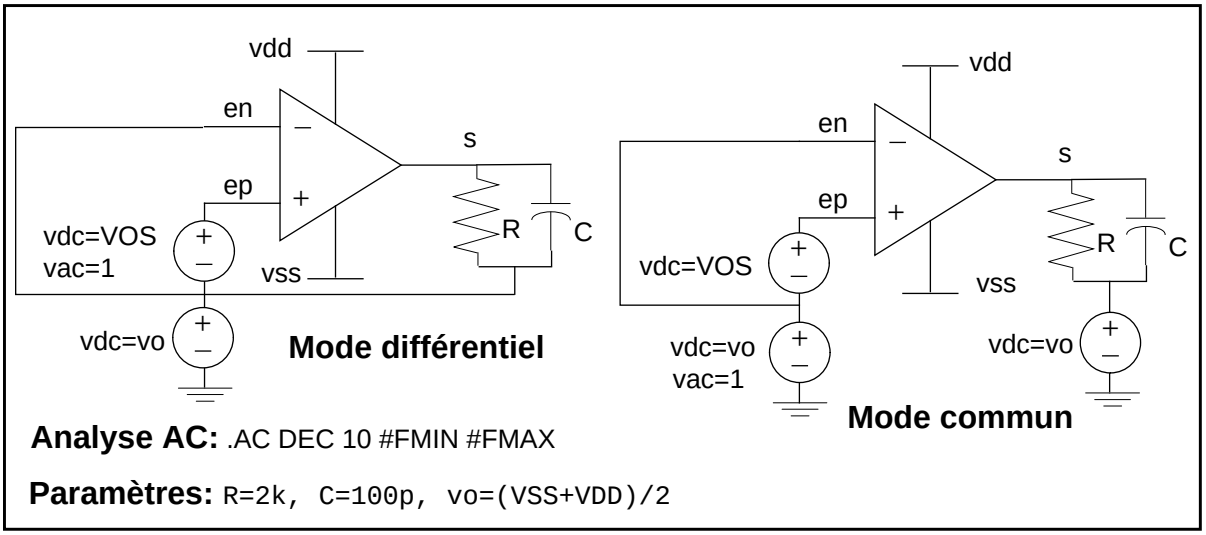


Figure 4 Schéma de la procédure AC_test

5 Courants de sortie (ICC_test)

Dans le schéma de cette procédure (cf Figure 5), les amplificateurs sont montés en suiveur: la tension de sortie est donc égale à la tension d'entrée pour une charge élevée. Lorsque la charge diminue pour s'approcher des conditions de court-circuit, la tension V(s) chute progressivement (en valeur absolue). On définit donc le courant *Isource* (resp. *Isink*) comme la valeur de la source *iout* pour laquelle la tension de sortie *v(s)* dévie de la tension positive *V1* (resp. négative *V2*) d'un certain pourcentage.

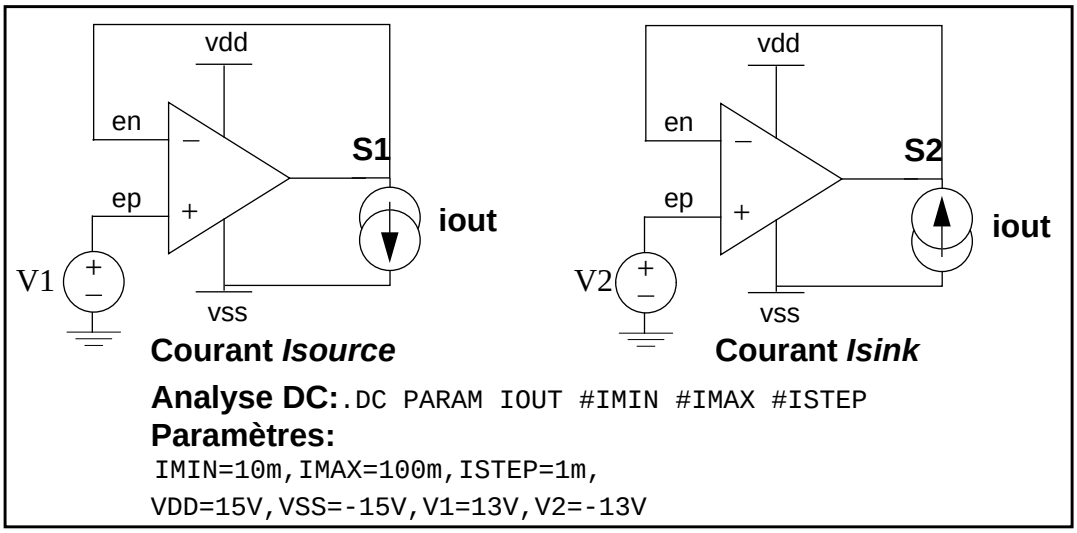


Figure 5 Schéma de la procédure ICC_test

Cette même procédure permet d'autre part de mesurer la résistance de sortie de l'op-amp, lorsque les tensions d'entrée, V1 et V2, sont proches des tensions d'alimentation. En effet, dans le cas du schéma mesurant *Isource*, alors que le courant de charge augmente progressivement de 0 jusqu'à *Isource*, la tension de sortie reste tout d'abord égale à la tension d'entrée puis diminue progressivement selon la loi $V_{out} = V_{max} - R_{out} \cdot I_{out}$, où V_{max} est la tension maximale de l'op-amp et R_{out} la résistance de sortie. Ceci est explicité par la Figure 6. R_{out} est ici mesurée au point d'intersection (Io,V1) suivant l'expression $R_{out} = (V_{max} - V_1) / I_0$.

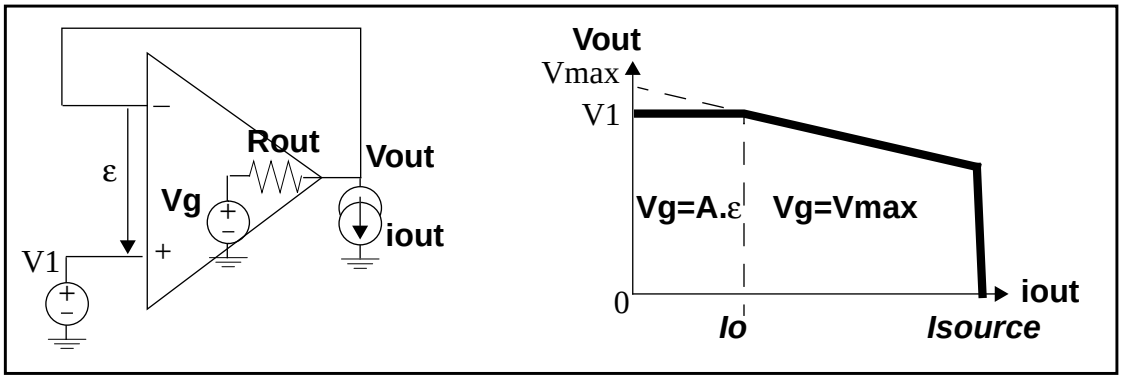


Figure 6 Tension de sortie de l'op-amp lors de la mesure de *Isource*

6 Impédance de sortie (ROUT_test)

L'impédance de sortie est définie par la relation $Z(j\omega) = \frac{V(j\omega)}{I(j\omega)}$. Elle est obtenue en appliquant une source de courant petit-signal en sortie de l'op-amp (cf Figure 7).

La résistance de sortie est égale à $R = \|Z(0)\|$ et la capacité à $C = 1/(2\pi f_{3dB} \cdot R)$ (modèle du premier ordre RC). Notons que pour le circuit étudié MC33272, la fréquence de coupure correspond au pôle dominant. Or, dans le cas des macros-modèles SPICE et HDL-A, ce pôle est modélisé dans l'étage de gain et l'étage de sortie ne modélise aucune racine. Les résultats de simulation sont donc dans ce cas très différents.

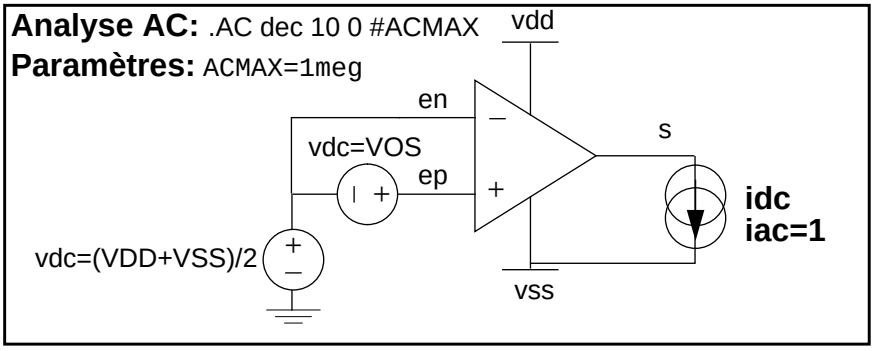


Figure 7 Schéma de la procédure ROUT_test

Notons que pour le circuit étudié MC33272, le résultat de cette mesure dépend fortement du courant continu *idc* qui est appliqué en sortie et qui détermine le point de fonctionnement de l'analyse.

7 Facteurs de réjection d'alimentation (SRR_test)

Le facteur de réjection positif (VPRR) est égal au gain DC de *V(S1)* en dB et le facteur de réjection négatif (VMRR) est égal au gain DC de *V(S2)* en dB (cf Figure 8).

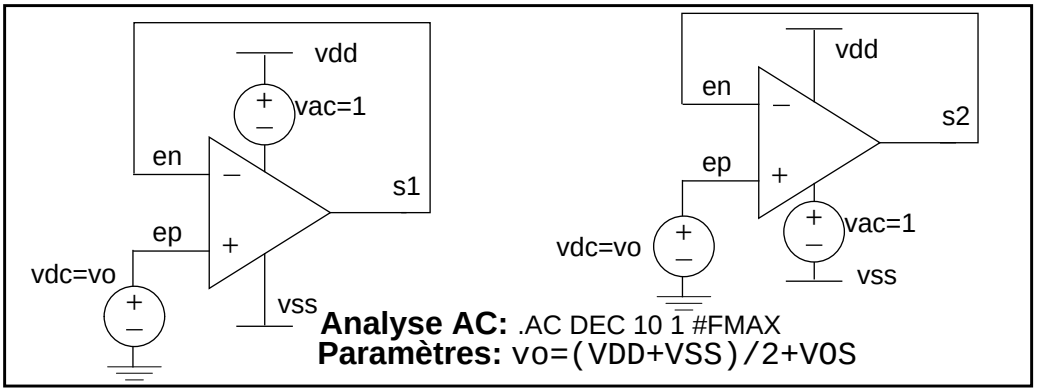


Figure 8 Schéma de la procédure SRR_test

Annexe 7: Traduction FAS vers HDL-A

Cette annexe présente le manuel utilisateur du traducteur FAS vers HDL-A, nommé FAS2HDLA, qui a été développé pour transférer le langage ELDO-FAS, initialement utilisé à SGS-Thomson, vers le langage de modélisation comportementale analogique/digital HDL-A. Le développement de ce langage s'est basé sur les définitions (de 1994) du futur standard IEEE 1076.1 VHDL-A et ouvre de nouvelles possibilités de modélisation.

FAS2HDLA Translator User's Guide

1 Introduction

FAS2HDLA is an automatic translator which is able to transfer ELDO-FAS analog behavioral models into HDL-A language. HDL-A has been designed on the basis of the current IEEE 1076.1 standardization work for analog extensions of VHDL and it can be considered as the first version of the future standard. Moreover, it is more powerful than FAS.

FAS2HDLA has already been successfully used to translate a library of around 30 FAS functional analog models into HDL-A. Minor manual modifications have been performed afterwards for model compilation, especially concerning variable types.

Finally, FAS2HDL-A is implemented with ZEBU tool, which is a powerful automatic generator of translators developed in SGS-THOMSON by P.Saramito.

The goal of this manual is to present the FAS2HDLA translation rules, about model structure, variable declaration and modeling statements.

2 User's guide

This translator makes use of an intermediate language named REF. The goal of this language is to ease the translation process which is decomposed into two translations: FAS2REF and REF2HDLA. FAS2HDLA is in fact a Shell script which is launched as the following:

% fas2hdla [-i] fas_model_file [-o hdla_model_file]

When specifying only **`fas_model_file`**, the source file will be **`fas_model_file.fas`** and the output one will be **`fas_model_file.hdla`**. Options -i and -o can be used to define whatever file names.

HDL-A models have to be compiled using the dedicated ANACAD compiler after specifying, and eventually creating, an HDL-A library name. They are instanciated in SPICE-like netlists as for FAS models but a .MODEL command is required for each HDL-A model type. Refer to "HDL-A User's Manual" for more information.

3 Model structure

FAS and HDL-A structures are very different as presented in Table 1. HDL-A is based on VHDL and is composed of an ***entity***, whose goal is to declare the interface of the model with the external world (parameters and connections), and of at least one ***architecture***, which describes the model behavior.

Notes:

Table 1 FAS/HDL-A structure (analog part)

	Fas model	HDL-A model

Interface & Declarations	`AMODEL modname (Pin List);`
	`interface declaration block`
	
	`local declaration block`
		`ENTITY modname IS`
		`interface declaration block`
		`END ENTITY modname ;`
		
		`ARCHITECTURE behavioral OF modname IS`
		`local declaration block`
		`BEGIN`
		`RELATION`

Initialization	`INITIALIZE`
	`initialization block`
	`ENDINITIALIZE`
		`PROCEDURAL FOR INIT =>`
		`  initialization block`

Analog description	`ANALOG`
	`IF MODE=DC THEN`
	`...`
	`ELSE`
	`  IF MODE=AC THEN`
	`  ...`
	`  ELSE`
	`  ...`
	`  ENDIF`
	`ENDIF`
	`ENDANALOG`
	`ENDMODEL`
		`PROCEDURAL FOR DC =>`
		`...`
		`PROCEDURAL FOR AC =>`
		`...`
		`PROCEDURAL FOR TRANSIENT =>`
		`...`
		`EQUATION(...) FOR DC =>`
		`...`
		`EQUATION(...) FOR AC =>`
		`...`
		`EQUATION(...) FOR TRANSIENT =>`
		`...`
		`END RELATION ;`
		`END ARCHITECTURE behavioral ;`

- FAS declarations are sorted by FAS2HDLA to identify interface declarations ("param", "ic" and "pin") and local ones ("local", "state", "istate","usep").
- HDL-A is an analog/digital language. The architecture can contain one digital part named ***process*** and one analog one named ***relation***. FAS2HDLA generates a ***process*** for sampling models.
- The analog ***relation*** block is composed of several sub-blocks: ***procedural*** blocks for initialization (***init***) and analysis modes (***dc***, ***transient*** , ***ac***) and ***equation*** blocks that are also associated to analysis modes. FAS2HDLA has therefore to analyze FAS conditional instructions in order to identify analysis modes. Table 1 shows one example of FAS analog description but whatever configurations are managed by FAS2HDLA.

4 Declaration

The translation of variables of basic types follows the rules given in Table 2. The declaration order **`GENERIC-COUPLING-PIN`** in the entity is required.

Table 2 Variable declaration

Variable	Fas model	HDL-A model
parameter	`DECLARE PARAM list_i: i-type;` `DECLARE PARAM list_j: j-type;`	`GENERIC ( list_i:i-type;list_j:` `  j-type ) ;`

Table 2 Variable declaration

Variable	Fas model	HDL-A model
coupling	`DECLARE IC list_i: ELECTRICAL;` `DECLARE IC list_j: ELECTRICAL;`	`COUPLING ( list_i,list_j: ANALOG );`
pin	`DECLARE PIN list_i: ELECTRICAL;` `DECLARE PIN list_j: ELECTRICAL;`	`PIN ( list_i,list_j: ELECTRICAL ) ;`
local	`DECLARE LOCAL list : type;`	`VARIABLE list : type;`
state	`DECLARE STATE list : type;`	`STATE list : ANALOG;`
implicit state	`DECLARE ISTATE list : type;`	`STATE list : ANALOG;`
constant	`DECLARE USEP list : type;`	`CONSTANT list : type;`

Notes:

- only *electrical pins* are managed by FAS2HDLA.
- only *electrical couplings (ic)* are managed by FAS2HDLA. FAS *ic* is translated into *analog coupling* although the HDL-A *coupling* notion is more general than FAS one.
- FAS and HDL-A local, constant and parameter variables can be boolean, integer or real.
- FAS analog variables, *state* and *istate*, are transferred into HDL-A *analog state* variables.

Vector declaration translation is presented in Table 3. FAS2HDLA manages only FAS *local*, *param* and *state* vectors.

Table 3 Vector declaration translation

FAS	HDL-A
`DECLARE type V(low:high) : REAL;`	`type V : REAL_VECTOR(low TO high);`
`DECLARE type V(low:high) : INTEGER;`	`type V : INTEGER_VECTOR(low TO high);`
`DECLARE type V(low:high) : BOOLEAN;`	`type V : BOOLEAN_VECTOR(low TO high);`
`DECLARE STATE V(low:high) : V_type;`	`STATE V : ANALOG_VECTOR(low TO high);`

5 Initialization

In HDL-A, the parameter (*generic*) initialization order must be the same as their declaration order in the entity. Constants must be initialized afterwards.
For this purpose, FAS instructions of the initialization block are sorted by FAS2HDLA (this is done by FAS2REF) and it is highly recommended to verify that the initialization will be carried out as wanted. Problems occur when you have in FAS the following declarations and initializations:

```
declare param a, b: real;
initialize
         make b = 1;
         make a = b;
endinitialize
```

FAS2HDLA generates the following part of HDL-A code, which is erroneous:

```
GENERIC ( a, b : REAL ) ;
...
PROCEDURAL FOR INIT =>
   a := b ;
   b := 1 ;
```

The user should change the order of FAS parameter declarations, such as:

```
declare param b, a: real;
```

Notes:

- expression types are not verified by FAS2HDLA and some type errors can be found at compilation. In the previous example, HDL-A instruction "b := 1" will be rejected because b is declared as a real. So you must change 1 by 1.0. A similar problem occurs with boolean variables: FAS boolean variables can be initialized to 0 or 1, while HDL-A boolean values are only FALSE or TRUE.
- initialization of FAS *max_tstep* variable is skipped by FAS2HDLA.
- initialization of FAS *sampling* variable is used to create a digital *process* which generates a periodic digital signal named CLOCK, of type BIT:

```
SIGNAL CLOCK : BIT ;
PROCESS BEGIN
       CLOCK <= '0', '1' AFTER TIME(val) ;
       -- val is the sampling value
LOOP WAIT ON CLOCK ;
       CLOCK <= NOT CLOCK AFTER TIME(val) ;
END LOOP ;
END PROCESS ;
```

6 Statements

6.1 Assignment statements

There are three types of assignments in an analog model:

- ***variable assignments*** are available for local, constant and state variables:
  FAS syntax "*make a = expr*" is translated into "*a := expr ;*" in HDL-A.
- ***pin assignments***: pins are characterized by their two fields: "across" and "through". For FAS2HDLA, "across" stands for voltage and "through" stands for current. Table 4 and Table 5 present the functions used to apply voltage or current contribution values on pins.

Table 4 pin voltage definition

FAS model	HDL-A model
`volt_limit(p1,min, max)`	`not translated`
`make volt.across(p1) = val`	`p1.V %= val ;`
`make volt.across(p1,p2) = val`	`[p1, p2].V %= val ;`
`make volt.across(p1)=volt.slope(vnew,tcom)`	`SLOPE(p1.V, vnew, 0.0, tcom);`
`make volt.across(p1)=volt.slope(vnew,tcom,td)`	`SLOPE(p1.V, vnew, td, tcom);`

Table 4 pin voltage definition

FAS model	HDL-A model
`make volt.across(p1)= volt.rise(vstart,vend,tcom)`	`SLOPE(p1.V, vnew, 0.0, tcom);`
`make volt.across(p1) = volt.rise(vstart,vnew,tcom,td)`	`SLOPE(p1.V, vnew, td, tcom);`
`make volt.across(p1) = volt.fall(vstart,vnew,tcom)`	`SLOPE(p1.V, vnew, 0.0, tcom);`
`make volt.across(p1) = volt.fall(vstart,vnew,tcom,td)`	`SLOPE(p1.V, vnew, td, tcom);`
`make volt.across(p1) = volt.sin(voff,vamp,freq,td,decay)`	`SINE(p1.V,voff,vamp,td,freq, decay);`
`make volt.across(p1) = volt.pulse(init,von,td,tr,tf,ton,per)`	`PULSE(p1.V, von,init,td,ton,per,tr,tf);`

Table 5 pin current contribution definition

FAS model	HDL-A model
`make curr.on(p1) = val`	`p1.I %= val ;`
`make curr.on(p1) = curr.slope(vnew,tcom)`	`SLOPE(p1.I, vnew, td, tcom);`
`make curr.on(p1) = curr.slope(vnew,tcom,td)`	`SLOPE(p1.I, vnew, td, tcom);`
`make curr.on(p1) = curr.rise(vstart,vend,tcom,td)`	`SLOPE(p1.I, vnew, td, tcom);`
`make curr.on(p1) = curr.fall(vstart,vnew,tcom,td)`	`SLOPE(p1.I, vnew, td, tcom);`
`make curr.on(p1) = curr.sin(voff,vamp,freq,td,phi)`	`SINE(p1.I,voff,vamp,td,freq,decay);`
`make curr.on(p1) =curr.pulse(init,von,td,tr,tf,ton,per)`	`PULSE(p1.I, von,init,td,ton,per,tr,tf);`

- ***implicit assignments:*** they are generally used to write differential equations. They are available for *istate* FAS variables and *state* HDL-A variables. In FAS, an implicit equation is described by a *solve (istate_var, expr)* statement. Such statements can be found wherever in the analog part.
  In HDL-A, implicit equations are gathered in *equation* blocks:
  EQUATION (state_list) FOR mode_list => ... where *state_list* is the list of HDL-A *state* variables that are to be solved and *mode_list* is the list of analysis modes for which the equation block can be used.
  FAS2HDLA creates one *equation* block for each mode, if required. When a FAS *solve* statement is found, the following steps are performed:
  - a new *state* variable named *expr_var* is created and added in the *state* declaration list (*var* suffixe is the *state* variable to be solved),
  - the *solve* statement is replaced by an explicit assignment of *expr_var*:
    expr_var := expr;

- *(expr_var) == 0.0;* is set in the proper *equation* block, depending of the analysis mode of the initial *solve* statement.

  The following example illustrates the process:

```
amodel resistor_100(p1,p2);
       declare pin p1,p2: electrical;
       declare istate ires: real;
analog
       solve(ires, volt.diff(p1,p2) - 100.0 * ires)
       make curr.on(p1) = ires
       make curr.on(p2) = -ires
endanalog
endmodel
```

  FAS2HDLA gives the following HDL-A code:

```
entity resistor_100 is
       pin(p1,p2: electrical);
end entity resistor_100;
architecture behavioral of resistor_100 is
       state ires: analog;
       state expr_ires: analog;
begin
relation
       procedural for dc =>
              expr_ires := [p1,p2].V - 100.0 * ires;
              p1.I %= ires;
              p2.I %= -ires;
       -- idem for ac and transient
       ...
       equation (ires) for dc =>
              (expr_ires) == 0.0;
       -- idem for ac and transient
       ...
end relation;
end architecture behavioral;
```

6.2 Control statements

Table 6 Conditional & loop statements

FAS model	HDL-A model

IF expr THEN
block
ENDIF
Note: no mode is specified in expr
	IF expr THEN
	block
	END IF ;

IF expr THEN
block1
ELSE
block2
ENDIF
Note: no mode is specified in expr
	IF expr THEN
	block1
	ELSE
	block2
	END IF ;

WHILE expr DO
block
ENDWHILE
	WHILE expr LOOP
	block
	END LOOP ;

FOR c=expr1 UPTO expr2 DO
block
ENDFOR
	FOR c IN expr1 TO expr2 LOOP
	block
	END LOOP ;

Table 6 Conditional & loop statements

FAS model	HDL-A model

`FOR c=expr1 DOWNTO expr2 DO`
`block`
`ENDFOR`
	`FOR c IN expr1 DOWNTO expr2 LOOP`
	`block`
	`END LOOP ;`

`FOR c=expr1 UPTO expr2 STEP expr3 DO`
`block`
`ENDFOR`
	Warning`: STEP value is skipped because this feature is not implemented in HDL-A`

6.3 Analysis mode definition

In the FAS model, analysis modes are defined by using classical conditional statements where the condition is a boolean expression based on FAS *MODE* variable, equal either to *DC* or *TRANS* or *AC*. An HDL-A model is composed of different new *PROCEDURAL* blocks that are associated to a list of analysis modes *DC*, *AC* or *TRANSIENT*.

In the current version, three *PROCEDURAL* blocks are created (if needed) for each analysis mode. This means that the complete structure of the FAS model will change. Statements belonging to different modes will be copied as many times as it is required. FAS2HDLA identifies if there are *MODE* definitions in the conditional boolean expression. If this is the case, all the statements of the conditional statement are copied in corresponding *PROCEDURAL* blocks. When no mode is defined in FAS, it is assumed that the behavioral description stands for all kinds of analysis.

Let’s consider the following FAS model:

```
amodel opa(inp,inm,outp);
   declare param gain,p1 : real;
   declare pin inp,inm,outp: electrical;
   declare usep tau : real;
   declare local vin : real;
   declare state vout,dvout: real;
initialize
   make gain  = 1.0e+5          (* open-loop gain              *)
   make p1    = 1.0e+2          (* dominant pole frequency     *)
   make tau   = 1/(6.28*p1)
   vsource(outp)
endinitialize
analog
   make vout = volt.value(outp)
   if(MODE = DC)then
          make dvout = 0.0
   else
          make dvout = state.dt(vout)
   endif
   make vin = gain*volt.diff(inp,inm)
   make volt.across(outp) = vin-tau*dvout
endanalog
endmodel
```

FAS2HDLA will generate the following code:

```
ENTITY opa IS
    GENERIC (gain,p1 : REAL) ;
    PIN (inp,inm,outp : ELECTRICAL) ;
```

```
END ENTITY opa;

ARCHITECTURE behavioral OF opa IS
    CONSTANT tau : REAL ;
    VARIABLE vin : REAL ;
    STATE vout,dvout : ANALOG ;

BEGIN
    RELATION
   PROCEDURAL FOR DC =>
          vout := outp.V;
          dvout := 0.0;
          vin := gain*[inp,inm].V;
          outp.V %= vin-tau*dvout;
   PROCEDURAL FOR TRANSIENT =>
          vout := outp.V;
          dvout := DDT(vout);
          vin := gain*[inp,inm].V;
          outp.V %= vin-tau*dvout;
   PROCEDURAL FOR AC =>
          vout := outp.V;
          dvout := DDT(vout);
          vin := gain*[inp,inm].V;
          outp.V %= vin-tau*dvout;
   PROCEDURAL FOR INIT =>
          gain := 1.0e+5;
          p1 := 1.0e+2;
          tau := 1/(6.28*p1);

    END RELATION ;
END ARCHITECTURE behavioral;
```

Notes:

- it is also possible to translate models, where analysis mode definitions are included in classical conditional statements, such as in the following FAS model:

```
amodel opa(inp,inm,outp);
declare param     gain,vos : real;
declare pin       inp,inm,outp: electrical;
initialize
      make gain  = 1.0e+5
      make vos  = 1.0e-6
      vsource(outp)
endinitialize
analog
if (level=1) then
      make volt.across(outp) = gain*volt.diff(inp,inm)
else
if (level=2) then
      if (mode=dc) then
            make dv=0
      else
            make vout=volt.value(outp)
            make dv=tau*state.dt(vout)
      endif
      make volt.across(outp) = gain*(volt.diff(inp,inm)+vos)-dv
endif
endif
endanalog
endmodel
```

For level one, the op-amp is ideal, with a finite gain. For level two, input offset voltage and one pole are modeled. This model is translated into:

```
ENTITY opa IS
       GENERIC (gain,vos : REAL) ;
       PIN (inp,inm,outp : ELECTRICAL) ;
END ENTITY opa;

ARCHITECTURE behavioral OF opa IS
BEGIN
RELATION
PROCEDURAL FOR DC =>
       IF (level=1) THEN
              outp.V %= gain*[inp,inm].V;
       ELSE
              IF (level=2) THEN
                     dv := 0;      -- this statement is valid for DC
                     outp.V %= gain*([inp,inm].V+vos)-dv;
              END IF;
       END IF;

PROCEDURAL FOR TRANSIENT =>
       IF (level=1) THEN
              outp.V %= gain*[inp,inm].V;
       ELSE
              IF (level=2) THEN
                     vout := outp.V;
                     dv := tau*DDT(vout);
                     -- this statement is valid for TRANSIENT and AC
                     outp.V %= gain*([inp,inm].V+vos)-dv;
              END IF;
       END IF;

-- PROCEDURAL FOR AC is the same as for TRANSIENT

PROCEDURAL FOR INIT =>
       gain := 1.0e+5;
       vos := 1.0e-6;
END RELATION ;
END ARCHITECTURE behavioral;
```

- ***Warning: If the boolean expression of a conditional statement is composed of some tests on the MODE variable and some other tests, the other tests are skipped by FAS2HDLA.***

6.4 Report statement

FAS **`REPORT THAT 'message'`** function is translated into HDL-A **`REPORT "message";`** FAS reporting function **`EXAMINE var`** has no traduction in HDL-A where a debugger is available.

7 Expressions

7.1 Functions on pin voltage

Table 7 shows the available functions that can be applied on *pins*.

Table 7 functions on pin voltage

FAS model	HDL-A model
`volt.value(p1)`	`p1.V`
`volt.diff(p1, p2)`	`[p1,p2].V`
`volt.last_value(p1)`	`PREVIOUS(p1.V)` **`Warning`**`: translated PREVIOUS(p1)`
`volt.dt(p1)`	`DDT(p1.V)` **`Warning`**`: translated PREVIOUS(p1)`
`volt.rising(p1,vth)`	`RISING(p1.V, vth)`
`volt.falling(p1,vth)`	`FALLING(p1.V, vth)`

7.2 Functions on *coupling*

In FAS, *coupling* connections are named *ic* and are associated to a physical domain. Only *electrical* one is considered by FAS2HDLA and therefore *ic* must transmit currents. In HDL-A, couplings are not attached to a physical domain and can transmit whatever analog data. Table 8 lists only the functions translated by FAS2HDLA.

Table 8 functions on FAS *ic*

FAS model	HDL-A model
`curr.value(ic1)`	`ic1`
`curr.last_value(ic1)`	`PREVIOUS(ic1)`
`curr.dt(ic1)`	`DDT(ic1)`

7.3 Analog functions

Functions that can be applied on analog ***state*** and ***istate*** variables are all presented in Table 9.

Table 9 state functions (s1: state, is1: istate)

FAS model	HDL-A model
`state.value(s1)`	`STATE(s1)`
`state.last_value(s1)`	`PREVIOUS(s1)`
`state.last_value(s1,delay)`	`PREVIOUS(s1,delay)`
`state.dt(s1)`	`DDT(s1)`
`state.integ(s1)`	`INTEG(s1)`

7.4 Simulator variables

They are used to describe time- or temperature- dependant models and also sampling

models. Table 10 shows the available variables for a translation.

Table 10 global variables

FAS model	HDL-A model
`time`	`CURRENT_TIME`
`tstep`	`TIME_STEP`
`temp`	`TEMPERATURE`
`temp_last`	`PREVIOUS_TEMPERATURE`
`sampling_time`	`EVENT(CLOCK)`

Notes:

- *sampling_time* is a boolean variable used to impose periodical time step to ELDO, the period being specified in the initialization part with *sampling* variable. FAS2HDLA generates a PROCESS which delivers a CLOCK signal whose period is the double value of *sampling*. Therefore there is an edge for each *sampling* time step. Such events are detected with EVENT() function.
- *mode* FAS variable is used to generate PROCEDURAL blocks.

7.5 Operators and mathematical functions

Table 11 operators & math. functions

	FAS Functions	HDL-A Functions
Unary operators	`- not`	`- NOT`

Binary operators	`* / + -`
	`< <= > >=`
	`= ^=`
	`and or`
	`power`
		`* / + -`
		`< <= > >=`
		`= /=`
		`AND OR`
		`**`

Mathematical Functions	`sin cos tan asin acos`
	`atan ln log expo`
	`sqr abs`
		`sin cos tan asin acos`
		`atan ln log exp`
		`sqrt abs`

Notes:

- after translation, it can be required to verify the types of the function arguments and mathematical or boolean expressions. Errors are detected at HDL-A compilation.

8 Example

A VCO model is used as example to show some of the possibilities of FAS2HDLA translator. The FAS description can be:

```
amodel vco (inp,inn,out);
 declare pin inp, inn, out : electrical ;
 declare param amplitude, f0, gain : real ;
 declare state w1, w0, phi1, t1, dw, k : real ;
```

```
 initialize
         make amplitude = 5.0
         make f0 = 100
         make gain = 10
         make w0 = 6.28*f0
         make k = 6.28*gain
         vsource(out)
 endinitialize

 analog
  if (mode=dc) then
         make w1 = k*(volt.value(inp) - volt.value(inn))
         make phi1=0.0
         make volt.across(out)=0.0
  else
         if (mode=trans) then
               make w1 = k*(volt.value(inp) - volt.value(inn))
               make dw = (time-tstep)*state.last_value(w1)-(time-tstep)*w1
               make phi1 = dw + state.last_value(phi1)
               make t1=(w0+w1)*time+phi1
               make volt.across(out)=amplitude*sin(t1)
         endif
  endif
 endanalog
endmodel
```

The HDL-A model, generated by FAS2HDLA translator, is as follows:

```
ENTITY vco IS
 GENERIC (amplitude,f0,gain : REAL) ;
 PIN (inp,inn,out_ : ELECTRICAL) ;

END ENTITY vco;

ARCHITECTURE behavioral OF vco IS
   STATE w1,w0,phi1,t1,dw,k : ANALOG ;

BEGIN
 RELATION
   PROCEDURAL FOR DC =>
         w1 := k*(inp.V-inn.V);
         phi1 := 0.0;
         out.V %= 0.0;
    PROCEDURAL FOR TRANSIENT =>
         w1 := k*(inp.V-inn.V);
         dw := (CURRENT_TIME-TIME_STEP)*PREVIOUS(w1)-
                                                (CURRENT_TIME-TIME_STEP)*w1;
         phi1 := dw+PREVIOUS(phi1);
         t1 := (w0+w1)*CURRENT_TIME+phi1;
         out_.V %= amplitude*SIN(t1);


   PROCEDURAL FOR INIT =>
         amplitude := 5.0;
         f0 := 100.0;
         gain := 10.0;
         w0 := 6.28*f0;
         k := 6.28*gain;

 END RELATION ;
END ARCHITECTURE behavioral;
```

Notes:

- The *out* pin name of the FAS model is automatically replaced by *out_*, because *out* is a keyword in HDL-A. It would be the same for following list of HDL-A keywords: OUT, in, IN, v, V, i, I.
- Manual modifications are required for *real* variables. For example, in the assignment `f0 := 100; f0` has been declared as a real. Therefore, it is necessary to replace `100` by `100.0`.

9 Limitations

- FAS2HDLA handles only ***electrical*** physical domain.
- variable ***types*** are not checked by FAS2HDLA. Therefore, manual modifications are generally required after HDL-A compilation, especially for *real* and *boolean* expressions. But HDL-A compiler is powerful and gives clear error messages.
- parameter initialization statements must be verified because of order change which is carried out by FAS2HDLA: the initialization order must be the declaration order.
- FAS2HDLA does not translate a *solve* statement whose unknown is an array element.

Annexe 8: Traduction FAS vers C-FAS

Cette annexe présente le manuel utilisateur du traducteur FAS vers CFAS, nommé FAS2CFAS, qui a été développé pour transférer le langage interprété ELDO-FAS vers le langage compilé ELDO-C-FAS, d’exécution plus rapide mais de plus bas-niveau.

FAS2CFAS Translator User's Guide

1 Introduction

The automatic FAS2CFAS translator allows one to translate analog behavioral models, written in the ELDO-FAS language, into models described with the ELDO-C-FAS language. The objective of this translation is to ***increase simulation speed***. C-FAS is the C interface of FAS and is a compiled language (as C), while FAS is only interpreted. In counterpart, FAS is more easy to use than C-FAS.

FAS2CFAS has been implemented with ZEBU tool, which is a powerful automatic generator of translators, developed in SGS-THOMSON by P.Saramito. The goal of this manual is to present the FAS2CFAS ***translation rules*** about model structure, variable declaration, variable initialization and model description.

2 User's guide

2.1 FAS2CFAS translation

There are two ways for launching FAS-->CFAS translation:

- %> **`fas2cfas < name1 > name2`**
  where **`name1`** is the name of the source file and **`name2`** the target name.
- **`%> fas2cfas name`** (without any suffix)
  where **`name`** is the base-name of source (**name.fas**) and target (**name.cfas**) files.

2.2 C-FAS compilation

C-FAS models have to be compiled and linked to an ELDO shared library, together with three files from ANACAD's installation, that you have to copy in your work directory (**bbuser.c**, **bbuserlib.c**, **bbcustom.c**). The new C-FAS model name has to be specified in the ***bbuser.c*** file in the **`add_fas_macro()`** function, as the following example:

```
extern int model_name();
new_fas_fnc("model_name", model_name);
```

2.3 C-FAS simulation

The C-FAS model is instantiated as a FAS model but it must be associated to an ELDO **`.model`** command, such as in this example:

```
Y1 opamp ep en out ic: vout param: gain=1e5 p1=100
```

```
.model opamp macro lang=c
```

The parameter **lang=c** indicates that it is a C-FAS model. Another **.model** command can be used in order to define a set of parameters available for several macro-models of the same name:

```
Y1 opamp ep1 en1 out1 ic: vout1 model=opa_typ
Y2 opamp ep2 en2 out2 ic: vout2 model=opa_typ
.model opamp macro lang=c
.model opa_typ modfas gain=1e5 p1=10
```

In this example, both op-amps have the same parameters defined in `opa_typ`.
Finally, ELDO must be launched with the `-lib` option to link dynamically the C-FAS library to the simulator, as for example:

```
eldo -lib CfasLibDir/
```

where `CfasLibDir/` is the directory containing ELDO shared library files for the dynamic link.

3 Model structure

The common structure of both languages is presented in Table 1. A CFAS model is a C function whose arguments are defined in the FASARGU C-preprocessor macro. It is split into different sections depending on the value of the simulator variable MODE:

- ALLOCATE for model parameter declaration and initialization with default values,
- INIT_TEMP for constant variable calculations,
- DC, TRANSIENT and AC for the different analysis modes. Note that the translator FAS2CFAS does not yet handle AC description, which needs functions of the C complex domain.

Table 1 FAS/CFAS model structure

	FAS model	CFAS model
Header	`amodel Name (PinList);`	`Name FASARGU {`

Declarations & Initializations	`Declaration List`
	`initialize`
	`...`
	`endinitialize`
		`if (MODE == ALLOCATE) ||`
		`(MODE == INIT_TEMP)`
		`{ .... }`

Analog body	`analog`
	`....`
	`endanalog`
	`endmodel`
		`if (MODE == TRANSIENT) ||`
		`(MODE == DC) {`
		`....`
		`}`
		`}`

4 Declaration and initialization

FAS models are described using parameters (*param*), analog quantities (*state* and *istate)*, constants (*usep)*, intermediate variables (*local)*, whose types are *real*, *integer* or *boolean*, and

analog connection points (*pin* and *ic*). Some simulator variables can be also used in the model. Finally mono-dimensional arrays are also handled by FAS2CFAS.

In C-FAS, parameters, analog variables and constants are elements of vectors: resp. *Param[]*, *State[]*, *Usep[]*, that are *double* typed. Connection points (*pin* and *ic*) are also denoted through indexes in functions. FAS2CFAS generates and manages automatically such variable indexes. Moreover, it uses C cast functions in the mathematical or boolean expressions in order to avoid type errors, depending on the FAS variable declaration.

4.1 Parameters

C-FAS parameter declaration and initialization statements generated by FAS2CFAS allows one to define a set of parameters which is common to several C-FAS models (cf Table 2). Parameter values of such models are stored in a common ELDO structure, named **`Model`**. Three sets of statements are concerned in C-FAS:

- **`define_equivalence_model(`**"name**`,"P`**i**`")`** statements, where *name* is FAS parameter name and *pi* is the corresponding **`Model`** field name,
- **`Init_Model_Param("P`**i**`",`**value**`)`** statements that define default values for **`Model`** parameters. Values defined in the FAS initialization part are used. But they could be changed.
- **`p[i]=Model->pi`** assignments are used in order to store **`Model`** parameter values in an intermediate array **`p[]`**.

If don't want to use the **`.model`** command, delete these statements in the model.
Finally, model parameter default values are defined using **`default_param(`**"name**`",`**value**`)`** function and parameter index assignment is carried-out with **`order_param(`**"name**`",`**index**`)`**.

4.2 Analog variables (*state* and *istate*)

`state` and implicit **`istate`** FAS variables correspond to elements of a **`State[]`** C-FAS array. The index is generated according to FAS declaration order, even when FAS arrays of state variables are declared. C pre-processor **`#define`** commands are used to generate an easier readable C-FAS model. After completing declarations, two kinds of C-FAS statements are used:

- **`ask_for_states(n`**`)` for the definition of the number of `state` and `istate` variables,
- **`assign_sate_name(index,"name")`** for name-index assignation, in order to be able to print or plot analog variable values during or after simulation.

Some other functions have to be initialized when using integration and delay functions or implicit equation on analog variables in the FAS model: **`Init_Integ_State(index)`**, **`Mem-State(index)`**, **`Init_Solve_State(index).`** Table 3 points out all these functions. In the case of the **`mem(1:8)`** FAS array, a single `#define` command for **`MEM`** is generated with an index of 2 (it is the 2nd declaration). **`MEM`** is in fact a base index in the C-FAS array of state variables: FAS **`mem(i)`** will correspond to **`State[MEM+i-1]`**.

4.3 Constants (*usep*)

Constants are stored in the C-FAS **`Usep[]`** array. As for the analog variables, an index is

Table 2 Parameter declaration and initialization

	FAS model	C-FAS model

Declaration
 	
and
 	
Initialization
	`declare param gain, voff: real;`
	
	`initialize`
	`  make gain = 1e10;`
	`  make voff = 1e-3;`
	`endinitialize`
		`#define GAIN1`
		`#define VOFF 2`
		`if (MODE == ALLOCATE) || (MODE == INIT_TEMP)`
		`{`
		`double p[2];`
		`if (Model != NULL){`
		`  /* there is a MODEL definition */`
		`  define_equivalence_model(“GAIN”,”P1”);`
		`  define_equivalence_model(“VOFF”,”P2”);`
		`  if (MODE == ALLOCATE) {`
		`  Init_Model_Param(“P1”,1e10);`
		`  Init_Model_Param(“P2”,1e-3);`
		`  }`
		`  p[0] = Model->p1;`
		`  p[1] = Model->p2;`
		`  }`
		`else { /* there is no MODEL definition */`
		`  p[0] = 1e10; /* default values */`
		`  p[1] = 1e-3;`
		`}`
		`default_param(“GAIN”, p[0]);`
		`default_param(“VOFF”, p[1]);`
		
		`order_param(“GAIN”, GAIN);`
		`order_param(“VOFF”,VOFF);`
		`}`

Table 3 Analog variables (state, istate)

	FAS model	C-FAS model

declaration	`declare state v: real;`
	`declare state mem(1:8): real`
	`declare istate w: real;`
		`#define V 1`
		`#define MEM 2`
		`#define W 10`
		`ask_for_states(10);`
		`assign_sate_name(V,”V”);`
		`assign_sate_name(MEM,”MEM”);`
		`assign_sate_name(W,”W”);`

initialization	`make v = 0.0`
	
	`history(v)`
		`Init_State(V, 0.0);`
		`Init_Integ_State(V);`
		`Init_Solve_State(W);`
		`MemState(V);`

generated for each **`usep`** and the dimension of the array **`Usep[]`** has to be defined. **`Init_Usep(c, val)`** is also used for initialization (cf Table 4).

Table 4 Constants (usep)

	FAS model	C-FAS model

declaration	`declare usep t1, hcmrr: real;`	`#define T11`
		`#define HCMRR 2`
		`ask_for_usep(2);`

Table 4 Constants (usep)

	FAS model	C-FAS model
initialization	`make t1 = expr;`	`Init_Usep(T1, expr);`

4.4 Intermediate variables (*local*)

They are used to hold intermediate values of expressions. Their types are `real`, `integer` or `boolean`. FAS2CFAS translates them respectively into `double`, `long` and `int`.

4.5 Connection points (*pin, ic*)

FAS connection points can be either **`pin`**, obeying to Kirchhoff's laws, or **`ic`**, for coupling definition. C-FAS `pin` and `ic` connections are referenced through an index, which corresponds to their position after `PIN:` and `IC:` keywords of the model instantiating line. For the FAS2CFAS translator, the `pin` order is defined by the `pin` list of the model header and the `ic` order depends of `ic` declaration order (cf Table 5).
FAS `vsource()` and `isource()` initialization statements are translated into `VStatus()` and `ISignal()`. If a single `pin` is specified as a voltage or current source, it is in fact assumed that the reference is the ground (node 0 in ELDO netlist) which corresponds to C-FAS keyword `GND`.

Table 5 Connection points (pin, ic)

	FAS model	C-FAS model

header	`amodel ex (in, vout, iout);`	`#define IN1`
		`#define VOUT 2`
		`#define IOUT 3`

declaration	`declare pin in, vout: electrical;`
	`declare pin iout: electrical;`
	`declare ic c2, c1: electrical;`
		`#define C2 1`
		`#define C1 2`

initialization	`vsource(vout)` `isource(iout)`	`VStatus(VOUT, GND);` `ISignal(IOUT,GND);`

4.6 Simulator variable initialization

Two simulator variables can be set in FAS:

- the largest allowable time step `max_tstep` which is translated into `Set_hmax(val);` `max_tstep` can be set in the analog description part but the CFAS function `Set_hmax` must be used only in ALLOCATE mode.
- the sampling time `sampling` which produces `set_sample_time(val);`

5 Model description

5.1 Functions on voltages

FAS2CFAS manages only the ***electrical*** domain. Therefore, only voltages can be read or applied on ***pins*** (Table 6). It is assumed that FAS trigger functions (`slope`, `rise`, `fall`,`sin`,`pulse`) are applied between two pins `p1`,`p2` with `make volt.across(p1,p2)=...`

Table 6 Voltage access and definition functions

FAS model	C-FAS model
`volt.value(p1)`	`VPin(P1)`
`volt.diff(p1, p2)`	`VDiff(P1, P2)`
`volt.last_value(p1)`	`P_VPin(P1)`
`volt.dt(p1)`	`dvpin_by_dt(P1)`
`volt.rising(p1,vth)`	`IFVRise(P1, GND, vth)`
`volt.falling(p1,vth)`	`IFVFAll(P1, GND, vth)`
`volt_limit(p1,min, max)`	`Volt_Limiter(P1,min, max);`
`make volt.across(p1) = val`	`VAcross(P1, GND, val);`
`make volt.across(p1,p2) = val`	`VAcross(P1, P2, val);`
`make ... volt.slope(vend,tcom)`	`VSlope(P1, P2, vend, tcom, 0.0);`
`make ... volt.slope(vend,tcom,td)`	`VSlope(P1, P2, vend, tcom, td);`
`make ... volt.rise(vstart,vend,tcom)`	`VRise(P1, P2, vstart, vend, tcom, 0.0);`
`make ... volt.rise(vstart,vend,tcom,td)`	`VRise(P1, P2, vstart, vend, tcom, td);`
`make ... volt.fall(vstart,vend,tcom)`	`VFall(P1, P2, vstart, vend, tcom, 0.0);`
`make ... volt.fall(vstart,vend,tcom,td)`	`VFall(P1, P2, vstart, vend, tcom, td);`
`make ... volt.sin(voff,vamp,freq,td,phi)`	`VSin(P1, P2, voff, vamp, freq, td, phi);`
`make ... volt.pulse(vstart,vend,td,tr,tf,pw,per)`	`VPulse(P1,P2,vstart,vend,td,tr,tf,pw,per);`

5.2 Functions on currents

As FAS2CFAS manages only the ***electrical*** domain, only currents can be applied on ***pins***. Current flowing into external voltage sources can be transmitted through ***ic*** connections, as well as ***istate*** variables that are computed in other models. In Table 7, *c1* is the name of an ic connection and *p1* is a pin name. It is assumed that `slope`, `rise`, `fall`,`sin`,`pulse` functions are applied on `p1` with `make curr.on(p1)=...`

Table 7 Operation on currents

FAS model	C-FAS model
curr.value(c1)	Current(C1)
curr.last_value(c1)	P_Current(C1)
curr.dt(c1)	di_by_dt(C1)
make curr.on(p1) = val	IPin[P1] = val
make ... curr.slope(vend,tcom,td)	ISlope(P1, GND, vend, tcom, td)
make ... curr.rise(vstart,vend,tcom)	IRise(P1, GND, vstart, vend, tcom, 0.0)
make ... curr.rise(vstart,vend,tcom,td)	IRise(P1, GND, vstart, vend, tcom, td)
make ... curr.fall(vstart,vend,tcom)	IFall(P1, GND, vstart, vend, tcom, 0.0)
make ... curr.fall(vstart,vend,tcom,td)	IFall(P1, GND, vstart, vend, tcom, td)
make ... curr.sin(voff,vamp,freq,td,phi)	ISin(P1, GND, voff, vamp, freq, td, phi)
make ... curr.pulse(vstart,vend,td,tr,tf,pw,per)	IPulse(P1,GND,vstart,vend,td,tr,tf,pw,per)

5.3 Functions on analog variables

Table 8 Analog functions (s1: state, is1: istate)

FAS model	C-FAS model
s1	(type) State[S1] for R.H.S. State[S1] for L.H.S.
state.value(s1)	State[S1]
state.last_value(s1)	P_State(S1)
state.last_value(s1,delay)	DP_State(S1,delay)
state.dt(s1)	ds_by_dt(S1)
state.integ(s1)	Integ_Sate(S1)
solve(is1, expr)	Solve_State(IS1, expr);

Some special functions can be applied on these kinds of variables and they are all presented in the Table 8. The CFAS functions return `double` typed outputs.

5.4 Simulator variables

They are composed of time step control, temperature control and simulator mode

variables (cf Table 9). The largest allowable time step (`max_tstep` in FAS corresponding to `hmax` of ELDO) can be read and written in FAS while it can only be set in CFAS through `Set_hmax()`function.

Table 9 Simulator variables

FAS model	C-FAS model
`time`	`TIME`
`tstep`	`TSTEP`
`min_tstep`	`get_hmin()`
`epsilon`	`get_epsilon()`
`sampling_time`	`get_on_time()`
`temp`	`TempK()`
`temp_last`	`P_TempK()`
`mode (dc, trans, ac)`	`MODE (DC, TRANSIENT, AC)`

5.5 Conditional and loop statements

Table 10 Conditional & loop statements

FAS model	C-FAS model

`if expr then`
`  block`
`endif`
	`if (expr) {`
	`  block1`
	`}`

`if expr then`
`  block1`
`else`
`  block2`
`endif`
	`if ( expr ) {`
	`  block1`
	`else {`
	`  block2`
	`}`

`while expr do`
`block`
`endwhile`
	`while ( expr ) {`
	`  block`
	`}`

`for c=expr1 upto expr2 do`
`  block`
`endfor`
	`for (c=expr1; c<=expr2; c=c+1) {`
	`  block`
	`}`

`for c=expr1 downto expr2 do`
`  block`
`endfor`
	`for (c=expr1; c>=expr2; c=c-1) {`
	`  block`
	`}`

`for c=expr1 upto expr2 step expr3 do`
`  block`
`endfor`
	`for (c=expr1; c<=expr2; c=c+expr3) {`
	`  block`
	`}`

5.6 Mathematical functions

Note that Table 11 is concerned only by DC and TRANSIENT analysis modes. For AC

Table 11 Mathematical functions

	FAS Functions	C-FAS Functions
Unary operators	`- not`	`- !`

Binary operators	`* / + -`
	`< <= > >=`
	`= ^= and or`
	`power`
		`* / + -`
		`< <= > >=`
		`== ~= && ||`
		`pow(expr1, expr2)`

Mathematical Functions	`sin cos tan asin acos atan` `ln log expo sqr abs`	`sin cos tan asin acos atan` `ln log10 exp sqrt fabs`

mode, special CFAS functions have to be used in order to handle complex representations of analog variables. FAS2CFAS does not support them.

6 Outputs

Some functions are delivered by FAS to print out messages (`report that 'message'`) or variable values (`examine var`). They are translated in CFAS by using the `elprintf` command. It is also possible to exit the model with FAS `error ('message')` command which is translated into `elprintf("%s",message)` and `main_exit(-1)` commands.

7 Translation example

A VCO model is used as example to show some of the possibilities of FAS2CFAS translator. Let this FAS model:

```
amodel vco(inp,inn,out);
        declare pin inp, inn, out : electrical ;
        declare param amplitude, f0, gain : real ;
        declare state w1, w0, phi1, t1, dw, k : real ;
        initialize
             make amplitude = 5.0
             make f0 = 100
             make gain = 10
             make w0 = 6.28*f0
             make k = 6.28*gain
             vsource(out)
        ndinitialize
   analog
        if (mode=dc) then
             make w1 = k*(volt.value(inp) - volt.value(inn))
             make phi1=0.0
             make volt.across(out)=0.0
        else
        if (mode=trans) then
        make w1 = k*(volt.value(inp) - volt.value(inn))
        make dw = (time-tstep)*state.last_value(w1)-(time-tstep)*w1
        make phi1 = dw + state.last_value(phi1)
```

```
        make t1=(w0+w1)*time+phi1
        make volt.across(out)=amplitude*sin(t1)
        endif
        endif
   endanalog
endmodel
```

The CFAS model produced by *FAS2CFAS* is the following:

```
#include <stdio.h>
#include <math.h>
#include “bbfonc.h”
#include “typmod.h”

/* Pin */
#define INP 1
#define INN 2
#define OUT 3
/* Param */
#define AMPLITUDE 1
#define F0 2
#define GAIN 3
/* State */
#define W1 1
#define W0 2
#define PHI1 3
#define T1 4
#define DW 5
#define K 6
vco FASARGU
{
 if((MODE == ALLOCATE) || (MODE == INIT_TEMP))
 {
   double p[2];
   if (Model!=NULL) {
        define_equivalence_model(“GAIN”,”P3”);
        define_equivalence_model(“F0”,”P2”);
        define_equivalence_model(“AMPLITUDE”,”P1”);
        if (MODE==ALLOCATE) {
             Init_Model_Param(“P1”,5.0);
             Init_Model_Param(“P2”,100);
             Init_Model_Param(“P3”,10);
        }
        p[0]=Model->p1;
        p[1]=Model->p2;
        p[2]=Model->p3;
   }
   else {
        p[0]=5.0;
        p[1]=100;
        p[2]=10;
   }
   default_param(“AMPLITUDE”,p[0]);
```

```
  default_param(“F0”,p[1]);
  default_param(“GAIN”,p[2]);
  Init_State(W0,6.28*Param[F0]);
  Init_State(K,6.28*Param[GAIN]);
  VStatus(OUT, GND);
  order_param(“GAIN”,GAIN);
  order_param(“F0”,F0);
  order_param(“AMPLITUDE”,AMPLITUDE);
  ask_for_states(6);
  ask_for_usep(0);
  assign_state_name(K,”K”);
  assign_state_name(DW,”DW”);
  assign_state_name(T1,”T1”);
  assign_state_name(PHI1,”PHI1”);
  assign_state_name(W0,”W0”);
  assign_state_name(W1,”W1”);
 }
 if ((MODE == TRANSIENT) || (MODE == DC))
 {
  if ((MODE==DC)) {
       State[W1]=State[K]*(VPin(INP)-VPin(INN));
       State[PHI1]=0.0;
       VAcross(OUT, GND, 0.0);
  }
  else {
       if ((MODE==TRANSIENT)) {
            State[W1]=State[K]*(VPin(INP)-VPin(INN));
            State[DW]=(TIME-TSTEP)*P_State(W1)-(TIME-TSTEP)*State[W1];
            State[PHI1]=State[DW]+P_State(PHI1);
            State[T1]=(State[W0]+State[W1])*TIME+State[PHI1];
            VAcross(OUT, GND, Param[AMPLITUDE]*sin(State[T1]));
       }
  }
}
}
```

8 Limitations

- The current version of the FAS2CFAS does not take into account ***AC*** descriptions. The reason is that FAS AC statements have to be translated into the C complex domain, using dedicated C-FAS functions and procedures.
  For example, `a+b` expression would have to be replaced by `complex_add(&res,a,b)` with the following definition: `COMPLEX *res,a,b;`
- FAS2CFAS is limited to the ***electrical*** physical domain which is defined with ***volt*** across and ***curr*** through variables. Other domains are in fact available in FAS, such as mechanical or thermal or fluidic.

Annexe 9: Traduction FAS vers MAST

1 Introduction

Cette étude a débuté par une comparaison théorique des possibilités de modélisation analogique et digitale des deux langages, et a été suivie par l'élaboration d'un traducteur expérimental FAS vers MAST, limité aux fonctions analogiques. Les règles de traduction ont pu être ensuite validées par des simulations SABER.

D'autre part, deux versions de MAST (3.0 et 3.3) ont été successivement étudiées au cours de cette thèse. Bien que le traducteur ait été développé sur la base de la première, nous présentons aussi les caractéristiques de la nouvelle dont le but est de faciliter la modélisation.

2 Structure des modèles

Le Tableau 1 présente les différents blocs des modèles FAS et MAST.

Tableau 1 Structure comparée de FAS et MAST

<table>
<tr><th></th><th>FAS</th><th>MAST</th></tr>
<tr><td>Interface</td><td>AMODEL nom (pins) ;
- déclaration des variables</td><td rowspan="2">TEMPLATE nom pins,couplages = paramètres

- déclaration des connexions
- déclaration et initialisation des paramètres</td></tr>
<tr><td>Initialisations</td><td>INITIALIZE
- initialisation des paramètres & constantes
ENDINITIALIZE</td></tr>
<tr><td>Description</td><td>ANALOG
- équations explicites et implicites
ENDANALOG
ENDMODEL</td><td>{
- declaration des variables intermédiaires
- initialisation des constantes (parameters)
- sections analogiques (values, equations)
- sections digitales (when)
- section de contrôle (control_section)
}</td></tr>
</table>

Le Tableau 2 décrit l'utilisation des diverses sections MAST qui sont nécessaires à la version 3.0. Certaines sont optionnelles pour 3.3, à l'exception de la section *control_section* et

des sections digitales *when*.

Tableau 2 Les sections d'un modèle MAST

Section MAST	Utilisation
parameters{ }	- définition des constantes - exécutée une seule fois - vérification de la cohérence des valeurs des paramètres
when (condition) { }	- simulation digitale dirigée par les évènements - programmation et détection d'évènements digitaux - le bloc est exécuté lorsque la condition est satisfaite
values { }	- équations explicites: calculs intermédiaires
control_section { }	- informations spécifiques à SABER concernant la résolution des équations non-linéaires relatives à chaque variable indépendante, la description de sources de bruit, les conditions initiales...
equations{ }	- équations implicites - équations de connexion sur les bornes analogiques - fonctions de dérivation et de délai
netlist	- description structurelle hiérarchique

MAST est donc plus structuré que FAS. La traduction FAS-->MAST nécessite par conséquent un tri des instructions selon leur utilisation ou selon le type des variables...

3 Déclaration des variables

Le Tableau 3 présente les différents types de variables utilisées par FAS et MAST pour la modélisation analogique et digitale et leurs déclarations. Bien que FAS ne permette pas de manipuler des variables digitales, il possède certaines fonctions de description mixte (échantillonnage, génération de fronts, détection de franchissement de seuils) qui devront être traduites en utilisant des variables digitales de MAST. Les paragraphes suivants donneront des détails quant à leurs utilisations.

Notes:

- Les variables nommées ***state*** par les deux langages ne sont pas équivalentes. En MAST, ce type de variables est utilisé pour la simulation digitale par les fonctions de programmation/ détection d'évènements. Ce sont des variables qui appartiennent à la liste des évènements digitaux du simulateur mixte SABER. Notons que les modèles MAST peuvent posséder des ports digitaux qui sont déclarés de type *state*.

Les variables *state* du langage FAS sont des variables analogiques dont les valeurs sont conservées au cours de la simulation. Elles sont par exemple utilisées comme arguments des fonctions de dérivation *state.dt(id)*, d'intégration *state.integ(id*) et de délai *state.last_value(id)*.

Tableau 3 Déclaration des variables FAS et MAST

Variables		FAS	MAST
Interface	Paramètres	`declare PARAM id{,id}:`*`type`*`;` *`type`*`: real,integer,boolean`	`NUMBER id[=init]{,id[=init]}` `(variables réelles)`

	Couplages	`declare IC id{,id} :` *`type`*`;`
		`type``:electrical,mechanical,`
		`...`
			`REF` *`unit`* `id{,id}`
			`unit``: v,i,q,nu (no-unit)`

	Bornes analog.	`declare PIN id{,id} :` *`type`*`;`
		`type``:electrical,mechanical,`
		`...`
			`type` `id{,id}`
			`type``:electrical,mechanical,`
			`...`

	Bornes digitales	`Néant`	`STATE` *`type`* `id{,id}` *`type`*`: nu, logic_4,...`
Internes	Constantes	`declare USEP id{,id}:real;`	`NUMBER id[=init]{,id[=init]}`
	Locales	`declare LOCAL id{,id}:`*`type`*`;` *`type`*`: real,integer,boolean`	`VAL` *`unit`* `id{,id}` `(variables réelles)`
	Analog.	`declare STATE id{,id}:real;`	`VAL` *`unit`* `id{,id}`
	Implicites	`declare ISTATE id{,id}:real;`	`VAR` *`unit`* `id{,id}`
	Branches	*`Néant`*	`BRANCH id_i=`*`through`*`(p->m),` `id_v=`*`across`*`(p,m)`
	Digitales	`Néant`	`STATE` *`type`* `id{,id}` *`type`*`: nu,logic_4,time,...`

- Les variables MAST analogiques (***ref***, ***var, val***) possèdent une dimension, associée à une unité, par exemple, ***v*** pour la tension exprimée en *volt*, ***i*** pour le courant en *ampère* et ***q*** pour la charge en *coulomb*. Les unités sont uniquement utilisées pour affichage après simulation. Les variables sans dimension particulière sont désignées par ***nu*** (*no unit*)**.** Cette dernière est utilisée par le traducteur FAS-->MAST pour la déclaration des variables FAS locales, analogiques et implicites (resp. *local*, *state*, *istate*).
- MAST 3.3 permet d'autre part de définir le couple de variables, potentiel (*across*) et flux (*through*), qui sont associées à une branche du modèle (***branch***).

4 Modélisation analogique

4.1 Opérations sur les connexions

Tableau 4 Opérations sur les bornes analogiques

Opération	Domaine	FAS	MAST

Lecture des potentiels	électrique
	translation
	rotation
	thermique
	fluide
		`volt.value(x)`
		`vel.value(x)`
		`rotvel.value(x)`
		`temperature.value(x)`
		`press.value(x)`
			`v(x)`
			`s(x)`
			`t(x)`
			`tc(x)`
			`pa(x)`

Application de flux (*through*)	électrique	`make curr.on(x) = ...`	`i(x) += ...`

Application de potentiel (*across*)	électrique	`make volt.across(x) = vo`	`var i ix`
			`ix : v(x) = vo`
			`i(x) += ix`

		`make volt.across(x,y)=vo`	`var i ixy`
			`ixy : v(x,y) = vo`
			`i(x->y) = ixy`

			`branch ixy=i(x->y),`
			`vxy=v(x,y)`
			`vxy = vo`

Deux types de connexion sont disponibles pour transmettre des informations: les bornes analogiques et les couplages. Concernant les bornes analogiques (cf Tableau 4), les langages FAS et MAST ne permettent que la lecture de leurs potentiels, qui sont les variables dépendantes du système d'équations, les flux en étant les variables indépendantes. Ainsi, un modèle FAS ou MAST doit exprimer des relations de branche sous la forme $Flux = Admittance \times Potentiel$. Dans le cas des sources idéales de potentiel, pour lesquelles cette formulation n'est pas valable, FAS propose une instruction compacte (*volt.across*) alors que la description MAST requiert la définition du flux interne, à l'aide d'une équation *implicite*, du type `ix:v(x)=v0`, où `ix` est l'inconnue et `v0` le potentiel appliqué sur la borne `x`. Le flux ainsi défini est ensuite appliqué sur la borne considérée. Cependant, MAST n'accepte qu'une seule équation implicite par variable inconnue. Il faut donc utiliser une variable intermédiaire qui stocke l'expression du potentiel à appliquer dans chaque cas. Ceci sera explicité au paragraphe 4.3.1

En MAST 3.3, dans le cas où le potentiel est appliqué entre deux bornes, il est possible de déclarer les grandeurs *potentiel* (*across*) et *flux* (*through*) qui sont associées à la branche

considérée, par une instruction de *définition de branche*: `branch vxy=v(x,y),ixy=i(x,y)`. Il suffit ensuite d'affecter la valeur désirée à la variable `vxy` par une équation explicite.

Quant aux couplages, désignés *ic* en FAS et *ref* en MAST, les opérations autorisées sont différentes suivant le langage (cf Tableau 5). En FAS, un couplage n'est utilisé qu'en lecture, pour la description de modèles contrôlés par des flux (courants des sources de tension ou variables implicites *istate* définies dans d'autres modèles). De même, un modèle MAST peut faire référence à une variable implicite *var* définie dans un autre modèle. Mais l'équation de définition peut être en plus modifiée à partir d'un autre modèle, par ajout d'un terme (opérateurs `+=` et `-=`).

Tableau 5 Opérations sur les couplages

Opération	Domaine	FAS	MAST

Lecture	électrique
	translation
	rotation
	thermique
	fluide
		`curr.value(x)`
		`force.value(x)`
		`torque.value(x)`
		`qflow.value(x)`
		`flow.value(x)`
			`x`
			`x`
			`x`
			`x`
			`x`

Modification de l'équation de définition		`Néant`	`x += ...` `x -= ...`

4.2 Exemples de sources contrôlées

Le Tableau 6 présente les modèles FAS et MAST d'une source idéale de courant contrôlée par un courant (CCCS). Notons la concision de la description MAST 3.3. Le Tableau

Tableau 6 Source CCCS

FAS	MAST 3.0	MAST 3.3

`amodel cccs (bp,bn);`
`declare pin bp, bn:`
`electrical;`
`declare ic ie: electrical;`
`declare param k: real;`
`declare local iout: real;`
`initialize`
`  make k= 5`
`endinitialize`
`amodel`
`make iout=k*curr.value(ie)`
`make curr.on(bp)=iout`
`make curr.on(bn)=-iout`
`endmodel`
	`template cccs ie bp bn`
	`= k`
	`electrical bp, bn`
	`ref i ie`
	`number k = 5`
	`{`
	`  val i iout`
	`  values{`
	`  iout = k * ie`
	`}`
	`equations{`
	`  i(bp) += iout`
	`  i(bn) -= iout }`
	`}`
		`template cccs ie bp bn =k`
		`electrical bp, bn`
		`ref i ie`
		`number k = 5`
		`{`
		`branch iout=i(bp->bn)`
		`iout = k * ie`
		`}`

7 présente les modèles FAS et MAST d'une source idéale de tension contrôlée par une tension (VCVS).

Tableau 7 Source VCVS

FAS	MAST 3.0	MAST 3.3

amodel vcvs (ap,an,bp,bn);
declare pin ap, an, bp, bn:
electrical;
declare param k: real;
declare local vin : real;
initialize
  make k= 5
  vsource (bp,bn)
  (* source de tension *)
endinitialize
amodel
make vin = volt.diff(ap,an)
make volt.across(bp,bn)=
  k*vin
endmodel
	template vcvs ap an bp
	bn = k
	electrical ap,an,bp,bn
	number k = 5
	{
	val v vin, vout
	var i iout
	values{
	  vin = v(ap)-v(an)
	  vout= v(bp)-v(bn)
	}
	equations{
	  iout : vout = k * vin
	  i(bp) += iout
	  i(bn) -= iout }
	}
		template vcvs ap an bp bn
		= k
		electrical ap,an,bp,bn
		number k = 5
		{
		branch iout=i(bp->bn)
		branch vout=v(bp,bn)
		branch vin=v(ap,an)
		
		vout = k * vin
		}

4.3 Opérations sur les grandeurs analogiques

Nous n'aborderons ici que les instructions spécifiques à la modélisation comportementale analogique, telles que les équations implicites, les fonctions de dérivation, d'intégration et d'accès aux valeurs précédentes (cf Tableau 8).

Tableau 8 Traduction des opérateurs analogiques

Fonction	FAS	MAST
Implicite	solve(id, expr)	val nu expr_id values{ expr_id = expr } equations{ id: expr_id = 0 }
Dérivation	make d = state.dt(id)	var nu ddt_id values { d = ddt_id } equations{ ddt_id: ddt_id =d_by_dt(id) }

Intégration	make i = state.integ(id)	var nu i
		val nu expr
		values{ expr = i-ddt_i}
		equations{
		  ddt_i: ddt_i= d_by_dt(id)
		  i: expr=0
		}

Tableau 8 Traduction des opérateurs analogiques

Fonction	FAS	MAST

Délai	(* section *initialize:* *)
	history(id)
	(* section *analog:* *)
	make old=
	state.last_value(id,td)
		values{ old = last_id}
		equations{
		last_id: last_id= delay(id,td)
		}

	make old= state.last_value(id)	values{ old = last_id} equations{ last_id: last_id= delay(id, step_size) }

4.3.1 Équations implicites

Tableau 9 Modélisation d'un interrupteur

FAS	MAST (3.3)

amodel switch(p1,p2,c);
declare pin p1,p2,c:electrical;
declare param von,vo,io:real;
declare local v,vctrl:real;
declare istate i:real;
initialize
make von=0.5
make vo=1m
make io=1n
endinitialize
analog
make v=volt.diff(p1,p2)
make vctrl=volt.value(c)
if (vctrl>=von) then
solve(i, v-vo)
else
solve(i, i-io)
endif
make curr.on(p1)=i
make curr.on(p2)=-i
endanalog
endmodel
	template switch p1 p2 c = von , vo ,
	io
	
	electrical p1, p2, c
	number von = 0.5,
	vo = 1m,
	io = 1n
	
	{
	val nu vctrl, expr0
	branch i=i(p1->p2), v=v(p1,p2)
	
	vctrl = v(c)
	if (vctrl >= von){
	expr0 = v - vo
	}
	else {
	expr0 = i - io
	}
	i: expr0 = 0
	}

Les équations implicites permettent de calculer les variables FAS *istate* et les variables MAST *var*. Ce sont des variables indépendantes, résolues par le simulateur, et accessibles aux autres modèles par l'intermédiaire des connexions de couplage. À la différence de FAS, MAST ne permet de définir qu'une seule équation par variable implicite donnée. En MAST 3.0, les équations implicites doivent être regroupées dans la section *equations{}*, avec les équations de connexion. Dans le cadre d'une traduction automatique, il est en fait nécessaire de passer par une variable intermédiaire, associée à la variable inconnue, pour stocker l'expression à

résoudre dans chaque cas. Le Tableau 9 présente l'exemple d'un modèle d'interrupteur idéal, équivalent à une source de tension *vo* en position fermée et à une source de courant *io* en position ouverte. Le modèle MAST (forme 3.3) utilise la variable supplémentaire `expr0` pour stocker `v-vo` lorsque le switch est fermé et `i-io` lorsqu'il est ouvert.

4.3.2 Dérivation

Tableau 10 Modélisation d'un filtre du 2nd ordre

FAS	MAST

`amodel filter2(in,out);`
`declare pin in,out: electrical;`
`declare param a,b: real;`
`declare local vin,vout: real;`
`declare state dv1,dv2: real;`
`analog`
`  make vin=volt.value(in)`
`  make vout=volt.value(out)`
`  make dv1=state.dt(vout)`
`  make dv2=state.dt(dv1)`
 	
`  make volt.across(out)=vin-a*dv1-b*dv2`
`endanalog`
`endmodel`
	`template filter2 in out = a,b`
	`electrical in,out`
	`number a,b`
	`{`
	`var nu iout, deriv0, deriv1`
	`val v vin,vout,dv1,dv2`
	`  vin=v(in)`
	`  vout=v(out)`
	`  dv1=deriv0`
	`  dv2=deriv1`
	`  deriv0: deriv0 = d_by_dt(vout)`
	`  deriv1: deriv1 = d_by_dt(dv1)`
	`  iout: v(out) = vin-a*dv1-b*dv2`
	`  i(out)+=iout`
	`}`

L'opérateur de dérivation MAST, *d_by_dt* impose de même plusieurs contraintes: il doit être utilisé dans une équation implicite, il ne doit être ni précédé ni suivi de l'opérateur multiplicatif, et l'argument est une expression qui ne doit pas contenir d'opérateurs *d_by_dt*. Une variable implicite *var* est donc introduite pour le calcul de chaque dérivée pour remplacer l'opérateur FAS *state.dt()* dans les expressions, et une équation implicite est ajoutée. Cette méthode est illustrée par le modèle de la fonction de transfert du 2nd ordre $\frac{Vout}{Vin} = \frac{1}{1 + a \cdot s + b \cdot s^2}$, qui nécessite le calcul de la dérivée première et seconde de *Vout* (cf Tableau 10). Le modèle décrit en fait l'équation $V_{out} = V_{in} - a \times \frac{dV_{out}}{dt} - b \times \frac{d^2 V_{out}}{dt^2}$.

4.3.3 Intégration

MAST ne possèdant pas d'opérateur d'intégration, il faut différencier les équations intégrales de FAS, écrites avec la fonction *state.integ()*. Soit un modèle d'intégrateur, tel que $V_{out} = \int V_{in} dt$. Le modèle FAS et la description MAST correspondante sont décrits au Tableau 11. Les opérations de traduction à effectuer sont donc les suivantes:

Tableau 11 Modélisation d'un intégrateur

FAS	MAST

`amodel integre (in,out,gnd);`
`declare pin in,out,gnd:`
`electrical;`
`declare param vo: real;`
`declare state vin,vout:real;`
`initialize`
`make vo=0`
`endinitialize`
`analog`
`make vin = volt.diff(in,gnd)`
`if (mode = dc) then`
`make vout=vo`
`else`
`make vout=state.integ(vin)`
`endif`
`make volt.across(out,gnd)=vout`
`endanalog`
`endmodel`
	`template integre in out gnd = vo`
	`electrical in, out, gnd`
	`number vo=0`
	`{`
	`val nu expr0`
	`var nu deriv0, vout`
	
	`branch vin=v(in,gnd)`
	`branch ddp0=v(out,gnd)`
	`branch curr0=i(out->gnd)`
	
	`if (dc_domain){`
	`expr0=vout-vo`
	`}`
	`else {`
	`expr0=deriv0-vin`
	`}`
	
	`vout: expr0=0`
	`deriv0 : deriv0=d_by_dt(vout)`
	`ddp0 = vout`
	`}`

- application en sortie de la tension `vout`:

  ```
  branch ddp0=v(out,gnd), curr0=i(out->gnd)
  ddp0 = vout
  ```

  où `ddp0` et `curr0` sont respectivement le potentiel et le flux de la branche out->gnd.

- différenciation de l'équation `vout=state.integ(vin)`:

  ```
  vout: d_by_dt(vout) = vin
  ```

  où `vout` est l'inconnue. C'est une équation implicite qui doit être unique pour `vout`. Or dans cet exemple, il faut aussi tenir compte de l'équation d'initialisation DC `vout=vo`. Une variable analogique, `expr0`, est donc introduite pour stocker l'expression à résoudre:

  - `expr0=vout-vo` en mode DC,
  - `expr0=deriv0-vin` en mode transitoire et AC, où `deriv0` correspond à la dérivée de `vout` et est calculée par l'équation `deriv0: deriv0=d_by_dt(vout)`.

- définition de l'équation implicite `vout: expr=0` pour le calcul de `vout`.

Pour effectuer une telle traduction de manière automatique, il faudrait être capable d'analyser le modèle FAS en deux passes. En effet, le traducteur sait que la variable `vout` est une inconnue à partir du moment où il détecte l'instruction d'intégration. Il doit alors revenir en arrière et remplacer l'instruction `vout=vo` par `expr=vout-vo`. Il doit aussi modifier le type de vout, qui, de variable locale `val`, doit passer en variable analogique `var`.

4.3.4 Modélisation de délai

Tableau 12 Modélisation d'un buffer

FAS	MAST

amodel buffer(in,out);
declare pin in,out:electrical;
declare param td,a:real;
declare state vin:real;
initialize
vsource(out)
make td=1.0e-9
make a=1.0
endinitialize
analog
make vin=volt.value(in)
if (mode=dc) then
make volt.across(out)=a*vin
endif
if (mode=trans) then
make volt.across(out)=
a*state.last_value(vin,td)
endif
endanalog
endmodel
	template buffer in out = td, a
	electrical in, out
	number td = 1.0e-9, a = 1.0
	{
	val nu vin, volt0
	var nu curr0, vdelay0
	values{
	vin = v(in)
	if (dc_domain){
	volt0 = a*vin
	}
	if (time_domain){
	volt0 = a*vdelay0
	}
	}
	equations{
	i(out) += curr0
	curr0 : v(out)=volt0
	vdelay0 : vdelay0=delay(vin,td)
	}}

La modélisation de délai est effectuée en FAS par la fonction *state.last_value(id, td)* où id désigne une variable analogique et td l'expression du délai. Ce deuxième argument est optionnel et dans ce cas, la fonction retourne la valeur calculée au pas de temps précédent. Elle est de plus limitée aux analyses temporelles.

En MAST, un délai est modélisé par la fonction *delay(id, td)*, où *id* doit être une combinaison de variables systèmes (*var* ou *ref*). Outre les analyses temporelles, elle autorise aussi les analyses AC, ce qui est particulièrement utile pour l'étude des systèmes échantillonnés. Enfin, elle est utilisée de la même façon que la fonction de dérivation *d_by_dt*, c'est à dire dans les équations *système*. Notons que la valeur calculée au pas de temps précédent est obtenue en prenant pour *td* la variable du simulateur Saber *step_size*, correspondant au pas de temps de calcul. Le Tableau 12 décrit les modèles FAS et MAST d'un *buffer* de délai *td* et de gain *a*. Le modèle MAST résulte du traducteur FAS-->MAST.

5 Modélisation analogique/digitale

Bien que le langage FAS soit limité à la description analogique, la traduction en MAST des fonctions de détection de franchissement de seuil, ainsi que des fonctions de génération de

signaux (impulsion, front montant, front descendant), nécessite l'utilisation des fonctions digitales MAST. Celles-ci doivent être regroupées dans les sections *when( ) { }*. Bien que FAS2MAST soit limité au domaine purement analogique, le Tableau 13 donne les correspondances entre les fonctions FAS et MAST pour une traduction manuelle..

Tableau 13 Fonctions analogiques/digitales

<table>
<tr><th>Fonction</th><th>FAS</th><th>MAST</th></tr>
<tr><td>Détection
d'un seuil
en sens
ascendant</td><td><pre>if volt.rising(id, vth) then
...
endif</pre></td><td><pre>when(threshold(id, vth, before,
after)){
if (after > 0) { ... }
}</pre></td></tr>
<tr><td>Détection
d'un seuil
en sens
descendant</td><td><pre>if volt.falling(id, vth) then
...
endif</pre></td><td><pre>when(threshold(id, vth, before,
after)){
if (after < 0) { ... }
}</pre></td></tr>
<tr><td>Génération
d'un front
descendant
1ère solution FAS</td><td><pre>if ... and time > tend then
make tend=time+Td+Tcom
make volt.across(id) =
   volt.fall (V1, V0, Tcom,
Td)
endif</pre></td><td rowspan="2"><pre>when(...) {
   tstart=time+Td
   schedule_event(tstart,fall,1)
schedule_event(tstart+Tcom,
fall, 0)
}
values{
if (fall==1) {
v_id = V1+(V1-V0)*(time -
tstart)/Tcom
}
}
equations{
     i_id: v(id) = v_id
     i(id)+=i_id
}</pre></td></tr>
<tr><td>2nd solution FAS</td><td><pre>if ... then
   make tstart=time+Td
   make tend=tstart+Tcom
endif
if time>tstart and
          time<tend then
   make volt.across(id) =
V1+(V1-V0)*(time-tstart)/Tcom
endif</pre></td></tr>
</table>

La fonction MAST *threshold* permet de détecter des évènements survenant sur des variables digitales (*state*) et admet comme arguments une expression à tester, la valeur du seuil et optionellement deux variables digitales qui indiquent respectivement l'état précédent et l'état suivant. De plus, elle ne peut être utilisée que dans une section *when*. Cela signifie qu'il faut analyser l'expression de test des instructions conditionnelles FAS pour leur traduction.

Le Tableau 13 donne aussi l'exemple de la génération d'un front descendant de *V1* à *V0*, en un temps *Tcom* et après un délai *Td*. Pour cela, FAS propose une fonction *volt.fall*, qui possède cependant un inconvénient majeur: pour déclencher cette fonction, il faut s'assurer qu'aucune fonction de ce type est encore active sur la même borne. Une solution est d'effectuer un contrôle sur le temps de fin de déclenchement (*tend*) et attendre la fin du front d'un éventuel front précédent (*tend* doit être de type *state* pour mémorisation). Elle est donc

valable lorsque les temps de commutation du modèle sont plus petits que ceux des signaux d'entrée. La deuxième solution du Tableau 13 montre une description équivalente, sans cette fonctionEn MAST, une telle opération nécessite l'utilisation de la fonction de programmation d'évènements, *schedule_event*, qui est appliquée sur une variable digitale *state*, et qui doit être appelée d'une section *when*. Dans l'exemple précédent, la variable *fall* indique l'occurence du front et l'affectation de la tension de *id* est réalisée dans la section *values*. La génération d'un front ascendant serait effectuée de la même façon (fonction FAS *volt.rise*).

Prenons l'exemple d'un modèle de comparateur à hystérésis, ayant pour paramètres les niveaux de tension haut et bas, les seuils haut et bas, les temps de montée et de descente. Le code MAST est le suivant:

```
element template cmp ip in q = rt, ft, von, voff, oh,ol
electrical ip, in, q
number rt=1u, ft=1u, von=2, voff=1, oh=3.3, ol=0
{
number sr,sf
val     v    vin, vq
var     i    iq
state   nu   before, after, rise, flag, r[2]
state   time tstart, tend,
state   v    vlast
parameters{
   sr=(oh-ol)/rt # pente positive
   sf=-(oh-ol)/ft # pente négative
}
when (dc_init) { # Initialisation digitale
  if (vin>=von) {
    schedule_event(time, flag, 1) # Flag indique l'état logique
  }
  else {
    schedule_event(time, flag, 0)
  }
  schedule_event(time, rise, 0)
   tend=0
}
when(threshold(vin,von,before,after)) {           # Passage du seuil haut
   if (after>0) {                                 # Sens ascendant
        if (rise==-1) {      # une descente était déjà programmée
             deschedule(r)   # suppression de la programmation de rise
        }
        vlast=v(q)           # tension de départ
        tstart=time
        tend=time+(oh-vlast)/sr
        # Nouvelle séquence du signal rise: tstart-> 1, tend-> 0
        schedule_event(tstart,rise,1)        # début de la rampe ascendante
        r=schedule_event(tend,rise,0)        # fin de la rampe
        schedule_event(tstart,flag,1)        # état logique à 1
```

```
   }
}
when(threshold(vin,voff,before,after)) {          # Passage du seuil bas
   if (after<0) {                                 # Sens descendant
        if (rise==1) {       # une montée était déjà programmée
             deschedule(r)   # suppression de la programmation de rise
        }
        vlast=v(q)           # tension de départ
        tstart=time
        tend=tstart+(ol-vlast)/sf
        # Nouvelle séquence du signal rise: tstart-> -1, tend-> 0
        schedule_event(tstart,rise,-1)       # début de la rampe descendante
        r=schedule_event(tend,rise,0)        # fin de la rampe
        schedule_event(tstart,flag,0)        # état logique à 0
   }
}
when(event_on(rise)) {
   schedule_next_time(time)
}
values{
   vin = v(ip)-v(in)
   if (rise==0) {                 # état haut ou bas selon flag
        vq = flag*(oh-ol)+ol
   }
   else if (rise==1) {            # rampe ascendante
        vq = sr*(time-tstart)+vlast
     }
     else if (rise==-1) {         # rampe descendante
        vq = sf*(time-tstart)+vlast
     }
}
equations{
   iq: v(q) = vq                  # source de tension q
   i(q) += iq
}
}
```

On se reportera au manuel de référence de MAST pour une description détaillée des fonctions digitales. Disons simplement qu'une ***liste d'évènements*** est définie pour les signaux digitaux et qu'il est possible d'y ajouter des évènements futurs (`schedule_event`) ou d'en supprimer (`deschedule`). Il est de plus nécessaire de ***synchroniser*** les pas de temps analogiques avec les évènements digitaux. Ceci est réalisé par la fonction `event_on`, appliquée sur un signal digital, en association avec la fonction `schedule_next_time(time)`, qui impose un point de temps de simulation à l'algorithme analogique.

6 Traduction automatique FAS vers MAST

Tableau 14 Possibilités du traducteur FAS-->MAST

		FAS	MAST

Variables	Bornes
	Couplage
	Paramètres
	Constantes
	Analogiques
	Implicites
	Locales
		`pin electrical`
		`ic electrical`
		`param real`
		`usep real`
		`state real`
		`istate real`
		`local real`
			`electrical`
			`ref`
			`number`
			`number`
			`val`
			`var`
			`val`

	Simulateur	`mode=dc` `mode=trans`	`dc_domain` `time_domain`
Fonctions	Lecture de potentiel	`volt.value(x)` `volt.diff(p,m)`	`v(x)` `v(p) - v(m)`
	Lecture des couplages	`curr.value(c)`	`c`

	Application d'un potentiel	`volt.across(p)=.vo`
		
		`volt.across(p1,p2)=vo`
			`curr: v(p) = vo`
			`i(p) += curr`
			`curr: v(p1)-v(p2)=vo`
			`i(p1->p2) += curr`

	Application d'un flux	`curr.on(x) = io`	`i(x) += io`
	Dérivation	`state.dt(id)`	`var: var = d_by_dt(id)`
	Délai	`state.last_value(id, td)`	`var: var=delay(id,td)`
	Op. unaires	`- not`	`- ~`

	Op. binaires	`+ - * / power`
		`and or < > >= =<`
		`= ^=`
			`+ - * / **`
			`& | < > >= =<`
			`== ~=`

	Mathématiques	`sin cos tan asin acos atan` `ln log expo sqr abs`	`sin cos tan asin acos atan` `ln log exp sqr abs`
Instructions	Équation explicite	`make state = expr`	`val = expr`
	Équation implicite	`solve(istate,expr)`	`var : expr = 0`
	Condition	`if then else`	`if( ) { } else { }`

Une traduction automatique de l'ensemble des fonctions FAS, et en particulier des fonctions de modélisation mixte, semble très difficile. C'est pourquoi, l'outil qui a été développé, FAS2MAST, reste limité à la ***traduction des fonctions analogiques, mathématiques et algorithmiques*** (cf Tableau 14).

Prenons l'exemple d'un modèle FAS d'amplificateur opérationnel différentiel, ayant

comme paramètres le gain (*gain*) , la tension de décalage ou *offset* (*voff*), la résistance d'entrée (*rin*), les deux pôles (*p1*,*p2*) et le taux de réjection en mode commun (*cmrr*). Sa description est:

```
amodel opamp(inp,inm,outp);
 declare pin inp,inm,outp: electrical;
 declare param gain,voff,p1,p2,rin,cmrr : real;
 declare usep t1,t2,hcmrr : real;
 declare local vin : real;
 declare state v2,dv1,dv2 : real;
 initialize
   make gain = 1.0e+5 (* open-loop gain *)
   make voff = 0.0e0 (* voltage offset *)
   make p1 = 1.0e+2 (* dominant pole frequency *)
   make p2 = 1.0e+6 (* second pole frequency *)
   make rin = 1.0e+7 (* internal resistance *)
   make cmrr = 1.0e-5
   make t1 = 1.0/(2.0*3.14*p1)+1.0/(2.0*3.14*p2)
   make t2 = 1.0/(4.0*3.14*p1)/(3.14*p2)
   make hcmrr = cmrr/2.0
   vsource(outp)
 endinitialize
 analog
   make v2 = volt.value(outp)
   if (mode = dc) then
        make dv1 = 0.0
        make dv2 = 0.0
   else
        make dv1 = state.dt(v2)
        make dv2 = state.dt(dv1)
   endif
   make vin = Gain*(volt.diff(inp,inm)+voff)+
                  hcmrr*(volt.value(inp)+volt.value(inm))
   make curr.on(inp) = (volt.diff(inp,inm)+voff)/rin
   make curr.on(inm) = -(volt.diff(inp,inm)+voff)/rin
   make volt.across(outp) = vin-t1*dv1-t2*dv2
 endanalog
endmodel
```

Le modèle MAST généré par le traducteur FAS2MAST est le suivant:

```
template opamp inp inm outp = gain , voff , p1 , p2 , rin , cmrr
electrical inp, inm, outp
number gain = 1.0e+5, voff = 0.0e0, p1 = 1.0e+2,
p2 = 1.0e+6, rin = 1.0e+7, cmrr = 1.0e-5

{
val nu vin, v2, dv1, dv2, curron0, curron1, volt0
var nu deriv0, deriv1, curr0
number t1, t2, hcmrr
parameters{
   t1 = 1.0 / (2.0 * 3.14 * p1) + 1.0 / (2.0 * 3.14 * p2)
   t2 = 1.0 / (4.0 * 3.14 * p1) / (3.14 * p2)
```

```
   hcmrr = cmrr / 2.0
 }
values{
   v2 = v(outp)
   if (dc_domain) {
        dv1 = 0.0
        dv2 = 0.0
   }
   else {
        dv1 = deriv0
        dv2 = deriv1
   }
   vin = gain * ((v(inp) - v(inm)) + voff) + hcmrr * (v(inp) + v(inm))
   curron0 = ((v(inp) - v(inm)) + voff) / rin
   curron1 = - ((v(inp) - v(inm)) + voff) / rin
   volt0 = vin - t1 * dv1 - t2 * dv2
}
equations{
   deriv0 : deriv0 = d_by_dt(v2)        # dérivée première
   deriv1 : deriv1 = d_by_dt(dv1)       # dérivée seconde
   i(inp) += curron0                    # courant d'entrée sur inp
   i(inm) += curron1                    # courant d'entrée sur inm
   i(outp) += curr0                     # courant de sortie sur outp
   curr0 : v(outp) = volt0
 }
}
```

7 Conclusion

Cette étude a permis d'analyser deux approches différentes en matière de modélisation comportementale analogique/digitale: celle de ANACAD à travers FAS et celle de ANALOGY à travers MAST. Ce travail a aussi permis de développer un traducteur automatique FAS vers MAST limité à une description FAS purement analogique.