REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE FERHAT ABBAS- SETIF-

MEMOIRE

Présenté à

LA FACULTE DES SCIENCES DE L'INGERNIEUR
DEPARTEMENT D'ELECTRONIQUE

Pour l'obtention du diplôme de

MAGISTER
ELECTRONIQUE

Option : Instrumentation

Par

Mr : MERABET MOHAMMED

Thème

Etude et application d'un simulateur VHDL A l'implémentation de modèles de circuits électroniques

Soutenu le : ... /... / 2009 devant le jury :

Mr : A. MERZOUKI	Pr. Université de SETIF	Président
Mr : N. AMARDJIA	M.C. Université de SETIF	Examinateur
Mr : N. BOUROUBA	M.C. Université de SETIF	Examinateur
Mr : N. BOUZIT	Pr. Université de SETIF	Rapporteur

REMERCIEMENTS

Je souhaiterais tout d'abord exprimer ma gratitude au professeur Naceredine BOUZIT, chef de département d'électronique de l'université de Setif, pour son encadrement constant et efficace tout au long de l'élaboration de ce travail au cours duquel j'ai pu apprécier ses qualités tant humaines que professionnelles. Je le remercie également pour ses encouragements et ses remarques savamment distillés à des moments où l'espoir d'y arriver s'amenuisait quelque peu.

J'adresse aussi mes remerciements à monsieur Abdel Aziz MERZOUKI, professeur à l'université de Setif, pour m'avoir fait l'honneur de présider le jury de ce mémoire.

Je suis tout particulièrement reconnaissant à Monsieur Nacereddine BOUROUBA, Maître de Conférence à l'université de Sétif, Pour son soutien, ses conseils scientifiques, et surtout son amitié qui m'ont permis de travailler dans les meilleures conditions et de garder de très bons souvenirs. Et de participer à l'amélioration de mes travaux par ses conseils judicieux et sa très grande pédagogie.

Mes remerciements s'adressent aussi chaleureusement à Monsieur Noureddine AMARDJIA, Maître de Conférence à l'université de Sétif, d'avoir accepté d'être un de mes deux examinateurs, et de s'être plongé dans ce sujet en émettant des remarques constructives qui m'ont permis d'améliorer mon travail.

Mes derniers remerciements vont évidemment aux personnes à qui je tiens le plus et qui m'ont soutenu tout au long des années de mes travaux et toutes celles qui les ont précédées. Merci donc à mes parents. Et bien sûr un grand merci à celle qui me supporte quotidiennement et qui m'a notamment donné les forces pour finaliser mon travail : mon épouse.

RESUME

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. Cela est aujourd'hui possible grâce à l'offre récente de puissants langages de modélisation comportementale analogique et mixte, tel que le langage VHDL-AMS.

L'objectif de ce travail est de faire implémenter de modèles fonctionnels qu'on élabore à partir d'un circuit électronique de base à savoir l'amplificateur inverseur dans un simulateur de type VHDL, tout en assurant un espace mémoire restreint et un temps de simulation réduit en tenant compte de la précision. La production de ces modèles a été faite par l'utilisation la technique d'approximation par segments linéaires (P.W.L) sur les caractéristiques de transfert, d'entrée et de sortie. Cette technique a permet d'obtenir un modèle électrique simplifié auquel est adjoint des équations linéaires décrivant la fonction du circuit. Un regain en temps de simulation est atteint par cette approche.

Mots-clés : VHDL-AMS, modélisation comportementale, simulation, amplificateur inverseur, modélisation de circuits analogiques.

TABLE DES MATIERES

Tables des matières

Chapitre III
MODELISATION COMPORTEMENTALE VHDL-AMS

Chapitre IV
IMPLEMENTATION VHDL-AMS

TABLE DES

FIGURES

Tables des figures

LISTE DES

TABLEAUX

Liste des tableaux

INTRODUCTION

GENERALE

Les progrès des dernières décennies dans le domaine de la microélectronique s'expliquent non seulement par le perfectionnement des procédés de fabrication des circuits intégrés, mais aussi par l'intervention de l'informatique à travers la Conception Assistée par Ordinateur . En effet, lors de la conception d'un circuit intégré, outre les principaux critères d'optimisation utilisés (surface de la puce, rapidité de fonctionnement, sécurité de fonctionnement, consommation...), le temps de conception lui-même est un paramètre très important. A mesure qu'augmente le nombre de transistors intégrés sur une puce, l'aide apportée par la CAO devient primordiale et la simulation du circuit avant fabrication incontournable. Or cette étape de simulation peut être considérablement ralentie en raison de la complexité croissante des circuits soumis aux logiciels de simulation [1]. La modélisation comportementale apporte une réponse à ce problème. Au lieu de décrire un circuit au niveau structurel (transistor par transistor), on le décrit au niveau comportemental en modélisant son comportement électrique externe.

L'avènement de langages de description du matériel pour les circuits analogiques et mixtes, tel que le langage VHDL-AMS (Very high speed integrated circuits Hardware Description Language for Analogue and Mixed Signal), offre maintenant les outils nécessaires au développement des nouvelles méthodologies de conception [2]. Le langage permet de décrire et modéliser, outre les comportements électriques, les comportements thermiques, mécaniques… ou d'autres définis par l'utilisateur. Ainsi il est possible de simuler le fonctionnement du système complet dans son environnement (thermique, radiatif…) et de vérifier qu'il est conforme aux spécifications initiales [1].

Une des voies possibles pour résoudre le problème est de mettre au point des modèles de haut niveau écrits en VHDL-AMS, en partant des modèles comportementaux. Une fois mis au point et validés, ces modèles sont intéressants en termes de portabilité, de flexibilité et de relative indépendance vis-à-vis de la technologie de fabrication [3]. La difficulté est qu'à l'heure actuelle les industriels n'ont pas le temps de consacrer des ressources suffisantes à l'élaboration de ces modèles VHDL-AMS car l'entreprise est risquée Concrètement, il manque la «glu» logicielle qui fait le lien entre des modèles opérationnels et des modèles d'abstraction plus élevée.

Dans ce cadre, Le présent travail met le point sur l'apport de la modélisation comportementale VHDL-AMS dans le développement des outils de conception. En particulier, grâce à la mise en place

de simulateurs VHDL sur lesquels les industriels peuvent évaluer l'intérêt de cette approche, en termes de méthodologie. D'ores et déjà des résultats ont été obtenus, notamment sur un amplificateur inverseur , pour lequel notre travail a montré une excellente adéquation entre le circuit réel et le modèle comportemental écrit en VHDL-AMS, avec l'avantage pour ces derniers d'avoir des temps de simulation jusqu'à quatre-vingt fois plus rapides. Parallèlement, une méthodologie d'extraction des paramètres de ce circuit basé sur l'approche P.W.L (piece-wice linear) , a été expérimentée avec succès, ces travaux peuvent préfigurer une méthode de génération fiable de modèles de haut niveau de systèmes analogiques.

Les travaux présentés dans ce mémoire apportent une contribution à la modélisation comportementale VHDL-AMS de circuits électroniques ainsi leur implémentation dans un simulateur VHDL.

Après une introduction générale, le travail est constitué de quatre chapitres dont nous esquissons une brève description dans les lignes suivantes :

- *Chapitre I*

On rencontre assez souvent de problèmes lors d'un processus de simulation . Pour localiser la ou les sources de ces problèmes, on doit d'abord comprendre la structure de l'outil utilisé qui est le simulateur avant de passer à l'analyse du circuit cible. A cet effet, dans ce chapitre nous tentons de donner un aperçu général sur les différents simulateurs électriques, leurs critères de choix, ainsi leurs modes d'analyse.

- *Chapitre II*

Au deuxième chapitre nous avons étudié le langage VHDL-AMS et nous avons essayé de le simplifier afin qu'il puisse être consulté par n'importe quel utilisateur.

- *Chapitre III*

Ce chapitre détaille les différents objectifs et avantages de la modélisation comportementale VHDL-AMS, les méthodologies hiérarchiques de conception descendante et ascendante, dans un premier temps.

La deuxième partie du chapitre sera consacrée à la présentation de la méthodologie de modélisation VHDL-AMS des circuits électroniques.

.

- *Chapitre IV*

Dans ce dernier chapitre, nous décrivons le cheminement qui nous a conduit à implémenter des modèles de circuits électroniques dans un simulateur à base de VHDL. Pour cela, quelles sont les méthodes à suivre, quel langage de modélisation et quel simulateur utiliser ? On s'aperçoit que le simulateur " Simplorer " peut répondre à ces questions, et que le langage VHDL-AMS, de part ses potentialités, répond bien à nos attentes. Ceci permettra de définir un modèle haut niveau simplifié, en VHDL-AMS lors d'un traitement d'un exemple de taille importante de transistors (18 transistors) : Un amplificateur inverseur dont les résultats sont ensuite discutés.

Enfin, une conclusion générale est donnée en fin du mémoire on citons quelques perspectives de ce travail.

Chapitre I

LE PROCESSUS DE SIMULATION

I.1. Introduction

La simulation est une méthode largement employée dans l'industrie électronique et microélectronique en particulier. En effet, la complexité des calculs manuels qui caractérise la prise en considération des divers phénomènes physiques régissant le fonctionnement des dispositifs rend le recours à la simulation une nécessité absolue voir inéluctable [9] .

Un simulateur électrique est un programme informatique, qui à partir de la description d'un circuit et de ces variables d'excitation permet de calculer n'importe quelle caractéristique ou variable électrique (tension, courant, impédance…), en importe quel endroit d'un circuit et quelles que soit les excitations appliquées [16]. Ce programme est capable d'analyser la topologie d'un circuit et d'y appliquer les lois fondamentales des réseaux électriques. Pour ce fait, il doit connaître le fonctionnement de chaque composant de circuit. En effet, il permet d'étudier le comportement et l'évolution du système.

Grâce à un simulateur électrique, on peut directement vérifier la conformité des résultats qui sont passés par l'élaboration du prototype. L'objectif des simulateurs électriques est d'essayer d'analyser des circuits de grande complexité (plusieurs milliers des éléments non linéaires).

La simulation de circuits analogiques complexes présente deux problèmes majeurs :

- Les temps de simulation sont très importants.
- On est confronté à des problèmes de convergence (pas de résultats).

La solution proposée consiste alors à diminuer l'effort de calcul de simulateur en remplaçant le circuit ou certaines de ses fonctions internes par des modèles équivalents reproduisant le plus fidèlement possible les performances à prendre en considération. La réalisation de ces modèles est l'objectif actuel de nombreux travaux de recherche [3].

I.2. Procédure interne d'un simulateur

D'une manière générale, les simulateurs se servent de modèles conçus à partir des équations qui représentent le comportement physique du composant [16]. La précision de ces

modèles dépend du nombre des paramètres et de la complexité des équations. Le principe de ces paramètres est basé sur des méthodes d'extraction mathématique à partir des courbes expérimentales qui nécessitent des mesures précises.

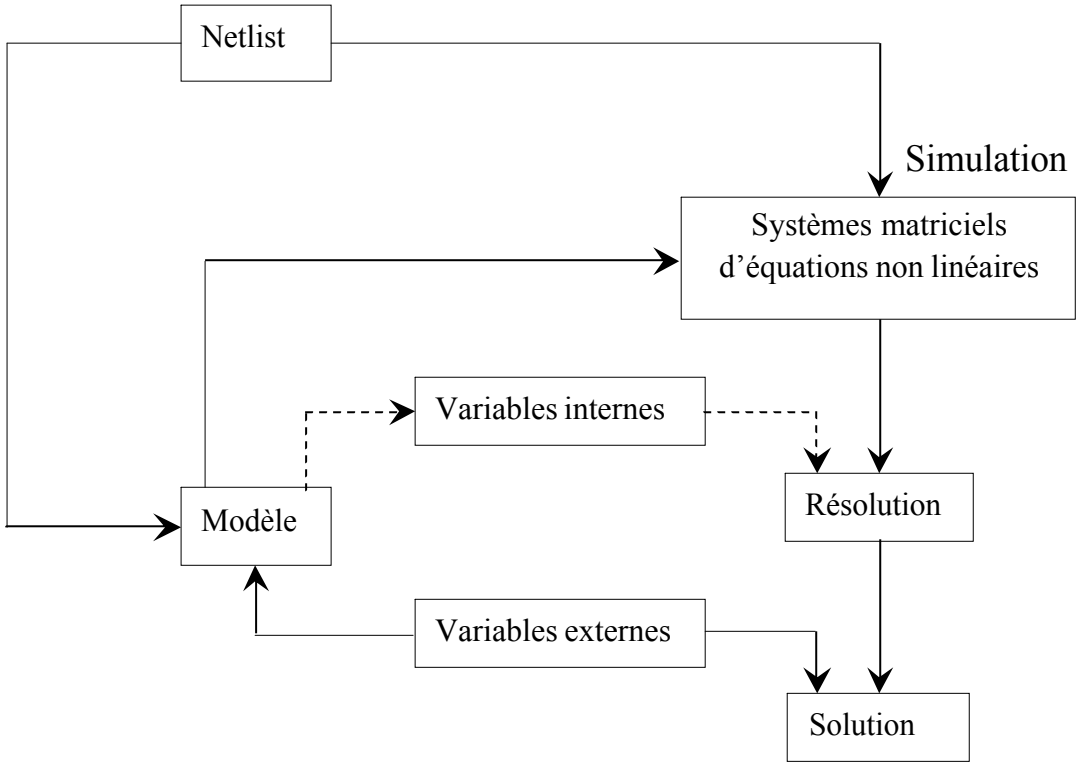


Figure.I.1 : Procédure interne d'un simulateur.

Les simulateurs sont destinés à l'analyse continue, fréquentielle et temporelle des circuits. Ils disposent pour cela l'algorithme de résolutions numériques des équations différentielles.

La figure I.1 décrit la procédure engagée lors d'une simulation, les circuits sont décrits dans un simulateur par une liste des interconnexions (***Netlis***t), qui indique comment sont connectés les éléments. A chaque modèle est associé un système d'équations qui décrivent les lois aux différents nœuds (tension et courant). Le simulateur résout ses systèmes d'équations non linéaires par des méthodes d'intégration numériques, des techniques itératives et des méthodes de résolution matricielle.

I.3. Techniques de simulation

I.3.1. Simulation logique [3] [10]

La simulation logique est une technique de simulation rapide basée sur l'évaluation de fonctions logiques et la propagation d'événements dans le modèle. Les signaux ne peuvent prendre qu'un nombre fini d'états et le temps est représenté par une valeur discrète, un multiple entier d'une unité de base appelée temps de résolution minimum (MRT - minimum resolvable time). Un événement est un changement de valeur sur un signal. La Figure I.2 donne l'algorithme général de la simulation dirigée par les événements (event-driven simulation).

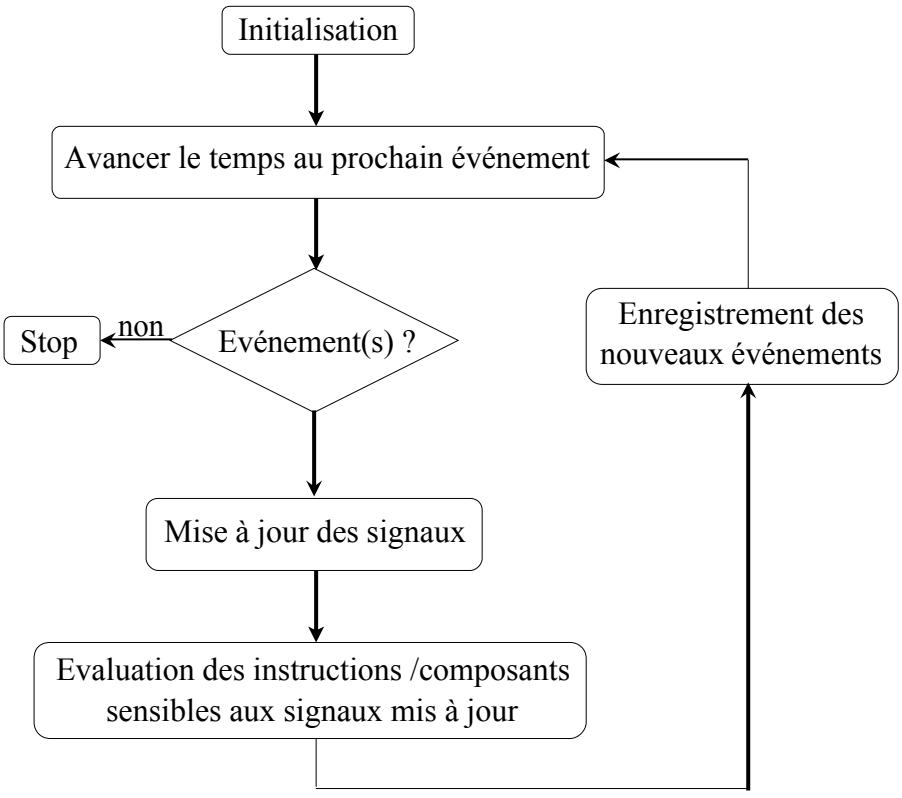


Figure.I.2 : Algorithme générale de simulation dirigée par événements.

La simulation démarre par l'affectation de valeurs initiales à tous les signaux. Il faut noter que l'état initial du modèle n'est pas nécessairement un état cohérent, ou stable car il n'y a pas encore de propagation des valeurs aux entrées primaires du modèle. Le temps de simulation est ensuite avancé jusqu'au moment du prochain événement prévu. Les valeurs des signaux ayant un événement à ce moment-là sont mises à jour et toutes les instructions ou les composants du modèle concernés par ces mise à jour sont réévalués. Ceci aboutit potentiellement à de nouveaux événements sur des signaux, au même instant ou à des temps futurs. La boucle se répète ainsi jusqu'à ce qu'il n'y ait plus d'événements à propager dans le modèle.

I.3.2. Simulation analogique [10]

La simulation analogique est beaucoup plus complexe que la simulation logique et requiert ainsi plus de ressources (temps de calcul, mémoire). L'archétype du simulateur analogique, ou électrique, est le programme SPICE, il a donné lieu à de nombreuses versions industrielles basées sur un même langage de description structurelle. 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 polynomiales (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.

La simulation analogique implique la résolution d'équations différentielles et algébriques linéaires et non linéaires. Les solutions sont des tensions entre les noeuds du circuit et les courants dans les branches du circuit. Normalement seulement un sous-ensemble de toutes les tensions et de tous les courants est requis.

La simulation analogique permet plusieurs types d'analyses :

- **L'analyse temporelle** (transient analysis) calcule les réponses temporelles du circuit (tensions et courants en fonction du temps) relativement à un ensemble de stimulus (sources et conditions initiales).

- **L'analyse DC** (direct current) calcule l'état du circuit pour un ensemble de stimulus fixes après un temps infiniment long (steady state). L'analyse DC est utile pour calculer le point de repos, ou de polarisation du circuit, des fonctions de transfert, la résistance d'entrée et de sortie du circuit, les sensibilités de variables de sortie en fonction de paramètres du circuit.

- **L'analyse AC** (alternative current) calcule les réponses fréquentielles du circuit en régime de petits signaux sinusoïdaux appliqués autour du point de repos du circuit. L'analyse AC est utile pour calculer des fonctions de transfert (p. ex. gain en tension, trans-impédances) en fonction de la fréquence et des conditions de polarisation du circuit. Elle est aussi utile pour analyser l'influence du bruit et déterminer les caractéristiques de distorsion du circuit.

I.3.3. La simulation mixte logique-analogique

Le tableau I.1 récapitule les caractéristiques principales de la simulation logique et de la simulation analogique.

Caractéristique	Simulation logique	Simulation analogique
Variables/inconnues	Signaux logiques	Tensions, courants, etc.
Valeurs des inconnues	Quantifiées (‘0’, ‘1’, ‘X’, ‘Z’, etc.)	Réelles
Calcul de l’état du circuit/modèle	Evaluation de fonctions logiques	Résolution d’équations différentielles algébriques non linéaires
Etat initial (t=0)	Pas nécessairement un état stable	Etat stable (point de repos DC) requis
Itération à un temps donné	Affectation de signaux avec délais nul (délai delta)	Résolution de systèmes non linéaires
Représentation du temps	Discret, multiple du MRT	Réel
Gestion du temps	Dirigé par événement	Continue avec pas d’intégration variable
Contrôle du temps temporel	Evénements sur les signaux	Erreur de troncature locale ou équivalente
Types d’analyses	temporelle	Temporelle, DC, AC

Tableau. I.1 : Récapitulation des caractéristiques de la simulation logique et analogique

Au vu de cette table, la simulation mixte logique-analogique doit résoudre les problèmes suivants: [10]

- **Conversions entre valeurs logiques et analogiques** : Il s’agit de définir des conversions qui aient un sens physique et qui n’aboutissent pas à une perte de précision ou à des non convergences durant la simulation. Il faut noter que ces conversions ne sont que des artefacts de la simulation mixte et ne constituent pas des composants physiques dans le circuit.
- **Etat initial (à t = 0) du circuit/modèle** : L’analyse temporelle en simulation analogique requiert le calcul d’un point de repos DC correspondant à une solution des équations du circuit. Si ce n’est pas le cas, la suite de l’analyse a de fortes chances de diverger ou au mieux

d'être incorrecte. La simulation logique n'est pas aussi sévère car l'avancement du temps mettra le circuit dans le bon état.

- **Gestion du temps** : Non seulement le temps est représenté différemment en simulation logique et en simulation analogique, mais il est géré selon des critères qui ne sont pas communs. Il s'agit donc de définir des points de synchronisation entre les deux échelles de temps de manière à prendre en compte correctement les interactions logiques-analogiques.

- **Support des analyses possibles en simulation analogique** : Seule l'analyse temporelle est réellement applicable de manière commune aux deux modes. Il s'agit donc de définir l'état de la partie logique lorsque l'on veut procéder à une analyse DC ou AC de la partie analogique. Une manière simple est de considérer la partie logique comme stable, c'est-à-dire que les signaux logiques agissant à l'interface logique-analogique doivent être considérés comme des sources constantes (après conversion des valeurs logiques en valeurs analogiques).

I.4. Les trois analyses basiques de la simulation électrique

Pour caractériser complètement un circuit électrique, au moins trois types d'analyse doivent pouvoir être réalisés par un simulateur électrique: une analyse statique, une analyse transitoire et une analyse alternative petit signal. [3] [11]

I.4.1. L'analyse statique

L'analyse statique est utilisée pour déterminer les points de fonctionnement (la polarisation et les points de repos correspondant) ainsi que les caractéristiques de transfert des circuits électriques linéaires et non linéaires. Pour ce faire, le vecteur **X** du système d'équations différentielles $F\left(X,\overline{X},t\right)=0$ étant nul, les composants réactifs sont neutralisés (les condensateurs sont remplacés par des circuits ouverts et les inductances sont remplacées par des courts-circuits) et toutes les sources indépendantes sont considérées comme statiques. Si l'opérateur **F** est linéaire la solution est directement déterminée par une méthode numérique de résolution d'un système d'équations linéaires, sinon le système est préalablement linéarisé par une méthode numérique d'analyse non linéaire.

I.4.2. L'analyse Transitoire

L'analyse transitoire est utilisée pour déterminer la réponse temporelle d'un circuit pour une durée d'observation [0, T]. Pour ce faire, l'intervalle [0, T] est discrétisé afin d'obtenir les instants de calcul [0, t_1, t_2,…, T]. Puis, partant d'un jeu de valeurs initiales spécifié par l'utilisateur ou par une analyse statique, pour chaque instant de calcul une intégration numérique est effectuée par une méthode numérique d'intégration afin de transformer le système d'équations différentielles en système d'équations algébriques. Enfin, le système est linéarisé par une méthode numérique d'analyse non linéaire pour délivrer la solution par une méthode numérique de résolution d'un système d'équations linéaires.

I.4.3. L'analyse alternative petit signal

L'analyse linéaire alternative petit signal est utilisée pour déterminer les caractéristiques fréquentielles d'un circuit. Pour ce faire, tous les éléments actifs non linéaires du circuit sont modélisés par un circuit équivalent linéaire petit signal autour d'un point de fonctionnement déterminé par une analyse statique. Tous les stimuli sont sinusoïdaux et de même fréquence mais peuvent avoir des phases relatives différentes. Les impédances ou les admittances sont mises sous leur forme opérationnelle et évaluées sur l'axe imaginaire du plan de la variable complexe P (P=jω). Ainsi, l'impédance d'un condensateur est considérée sous la forme opérationnelle $Z_C = \frac{1}{jc\omega}$, l'impédance d'une inductance est considérée sous la forme $Z_L = jL\omega$.

I.5. Evolution des simulateurs

La taille, la complexité et le degré de raffinement des modèles des dispositifs actifs utilisés, sont devenus tellement importants que la conception d'un circuit intégré est totalement impossible sans l'assistance d'outils informatiques de simulation [5]. En effet, ce n'est que par une évaluation rapide des performances d'un circuit sans avoir à le fabriquer, que l'on peut obtenir un temps de conception réduit entraînant une limitation des coûts de production. On distingue essentiellement trois catégories de simulateurs:

- La première catégorie est constituée des simulateurs logiques ou événementiels qui ne s'intéressent qu'au changement d'état logique des nœuds d'un circuit. Ils sont uniquement utilisés en électronique numérique pour évaluer fonctionnellement les performances. Sur chaque événement, le simulateur n'ayant à examiner que les éléments de circuit affectés par le changement d'état puis éventuellement à le propager, aucune résolution des lois de Kirchhoff et aucune analyse matricielle n'est requise. Ces outils sont ainsi très rapides et n'ont pratiquement aucune limitation en termes de taille et de complexité du circuit à simuler.

- La deuxième catégorie de simulateurs est également limitée à l'analyse des circuits numériques, elle est constituée des simulateurs temporels. Les dispositifs non linéaires sont remplacés par des modèles linéaires par morceaux ou tabulés, la quasi-unidirectionnalité et le fort taux d'inactivité caractérisant les "gros" circuits numériques (statistiquement plus de 80% du circuit est inactif sur une phase d'horloge) sont exploités. Ce type de simulateur "plus physique" que les simulateurs logiques permettent d'obtenir des précisions de l'ordre de quelques pourcent sur les temps de réponse pour un temps de calcul dix à cent fois plus long.

- Quant à la troisième catégorie, elle est constituée des simulateurs analogiques, ou simulateurs électriques, qui ont essentiellement pour vocation l'étude du comportement d'un circuit linéaire ou non linéaire sur un intervalle de temps continu (ou plus exactement discrétisé avec un pas de calcul théoriquement très inférieur à la plus petite constante de temps du circuit à analyser). [3]

La simulation logique et/ou temporelle et/ou analogique peut être combinée dans un même exécutable ou en fonctionnement séparé pour constituer un type de simulateur appelé simulateurs mixtes, permettant directement l'évaluation des performances électriques des systèmes électroniques mixtes constitués de circuits analogiques et numériques intégrés sur un même substrat, c'est à dire les **SoC** (**S**ystème **o**n **C**hip).

Enfin, des simulateurs analogiques dédiés ont également été développés pour des applications spécifiques, l'exemple type est le simulateur SWITCAP qui permet l'analyse des circuits à capacités commutées. Historiquement, les premiers simulateurs électriques sont contemporains des premiers ordinateurs puisque c'est dés 1950 qu'un calculateur composé de relais électromécaniques a été programmé pour l'analyse et la conception des filtres

électriques. Bien entendu leur évolution a suivi l'évolution des ordinateurs, et nous sommes passés en moins d'un quart de siècle d'un simulateur de première génération permettant difficilement la simulation de circuits composés de quelques transistors modélisés par une dizaine de paramètres à un outil de troisième génération permettant la simulation de circuits de plusieurs milliers de transistors MOS modélisés par une centaine de paramètres. [8]

Aujourd'hui, la plupart des simulateurs électriques en versions commercialisées ou versions publiques, sont des descendants du simulateur de deuxième génération **SPICE** (**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis)] développé et distribué gratuitement par l'université de Berkeley dès 1972.

I.6. Quelques simulateurs des circuits électriques

Il existe une gamme de simulateurs qui permet de simuler différents types de circuits électriques. Cette offre couvre essentiellement la simulation de circuits électroniques aussi bien analogiques que numériques ou mixtes. On rencontre également des logiciels dont la vocation est plus pédagogique, facile à utiliser, mais ne présentant pas un intérêt scientifique évident. [10]

Nom du logiciel	Développeur	Description
Pspice	OrCad	Simulateur Spice de circuits électroniques analogiques ou numériques **http : //www.orcad.com/products/pspice/**
Psim	Powersim	Simulateur de circuits électroniques de puissance et contrôle moteur **http://www.powersimtsch.com/**
Wincad	Mecrelic	Simulateur Spice de circuits électroniques analogiques ou numériques **http://glao.dezai.free.fr/**

Simplorer	Ansoft	Simulateur de circuits électroniques ou électromagnétiques **http://www.ansoft.com/products/em/simplorer/**
DXP	Protel	Simulateur de circuits électroniques analogiques ou numériques **http://www.protel.com/**
TKgate	Jeffery P.Hansen	Logiciel de simulation électronique gratuite, sources libres (licence GNU) **http//www-2.cs.cmu..edu/~Hansen/tkgate/**
Electric	Steven M. Rubin	Logiciel de simulation électronique et électrique créé en 1982 par Steven M.Rubin et développé à l'origine par Linux, sources libres (licence GNU) **http : //www.gnu.org/software/electric/**
Schemaplic	Fitec	Logiciel de conception et de simulation des schémas électriques **http://www.fitec.fr/interactif.htm**

Tableau I.2 : Quelques simulateurs de circuits électroniques

I.7. modélisation et simulation

La création d'un modèle résulte d'un processus de structuration d'un ensemble de connaissances, parfois expérimentales, que l'on dispose à propos d'un phénomène ou d'un système physique. Un modèle est une représentation abstraite d'une réalité physique dont on ne conserve que les aspects essentiels à une certaine utilisation, ou plutôt à une certaine expérimentation. Un modèle ignore donc délibérément des détails, soit parce qu'ils sont inutiles ou peu importants pour l'expérimentation en question, soit parce qu'ils ne sont pas connus.

Un modèle peut représenter ou décrire plusieurs facettes d'un phénomène ou d'un système:

• Le **comportement** exprime des relations de cause à effet, des algorithmes, des processus ou des équations définissant des relations entre des variables qui représentent des grandeurs physiques. Il peut aussi définir les états internes que le système peut prendre, les propriétés internes qu'il doit exhiber et les contraintes qu'il doit satisfaire. Un comportement complexe peut être décomposé en une hiérarchie de comportements plus simples.

• La **structure** décrit la manière dont un système est logiquement ou physiquement organisé en soussystèmes. Une structure logique (ou topologique) est adimensionnelle. Elle prend usuellement la forme d'un schéma, éventuellement hiérarchique. Une structure physique (ou géométrique) tient compte des dimensions, des tailles et des formes.

Le simulateur est l'un des outils essentiels d'aide à la conception assisté par ordinateur (CAO) dans l'industrie aujourd'hui, pour atteindre les objectifs spécifiés le plus vite et plus efficacement possible. C'est pourquoi le processus de la simulation demande trois ensembles de données et de programmes:

- Le moyen pour décrire le système à simuler (langage de description).
- La description du système (modèle).
- Le mécanisme de simulation du système qui a été conçu (simulateur).

I.8. Les différents types de langages

Nous pouvons distinguer principalement deux grandes familles de langages descriptifs: les langages de description logicielle et les langages de description matérielle appelés souvent **HDL** (Hardware Description Langage).

I.8.1. Les langages de description logicielle

Ce sont les langages de bas-niveau. Ces langages (C/ C++, FORTRAN,…) sont utilisés pour programmer des mini-simulateurs numériques ou pour vérifier certaines fonctionnalités.
Ces langages se caractérisent par leur souplesse et par la facilité de leur mise en œuvre dans certaines applications numériques.

Cependant, pour la modélisation et la simulation analogique, certains outils offrent des bibliothèques de fonctions, en générale codées en **C** , afin de créer de nouveaux modèles. Ces outils sont appelés simulateurs ouverts. C'est le cas du simulateur **ELDO** qui propose un ensemble de fonctions **C** dédiées à la modélisation analogique et appelé **CFAS** (C Function Analog Simulation), c'est le cas aussi du simulateur **Smartspice** de SILVACO. [28]

I.8.2. Les langages de description matérielle (HDL – Hardware Description Langages)

Le développement des langages de description matérielle a été fait dans un premier temps pour les circuits numériques. La nécessité de normaliser ces langages a donné naissance en 1987 à la norme IEEE 1076 définissant le VHDL (**V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage). Une extension en 1999 de cette norme a donné naissance à

la norme IEEE 1076.1 ou VHDL-AMS (VHDL for **A**nalog and **M**ixed **S**ystems), qui inclut la norme VHDL et qui permet de modéliser aussi des circuits analogiques et mixtes.

Parallèlement à la norme VHDL, nous pouvons aussi citer son concurrent le langage VERILOG, qui est plus utilisé outre Atlantique. [3] [28]

Un langage de description de matériel idéal a les propriétés suivantes: [10]

• Il supporte la description d'une large gamme de systèmes à la fois logiques (numériques) et analogiques. Pour les systèmes logiques, il supporte les systèmes combinatoires et séquentiels, synchrones et asynchrones. Pour les systèmes analogiques, il supporte non seulement des systèmes électriques, mais aussi mécaniques, thermiques, acoustiques, etc. Ce dernier aspect est très important pour la conception de systèmes car il s'agit de prendre en

compte les interfaces avec l'environnement extérieur dans lequel le système conçu sera testé et dans lequel il fonctionnera.

• Il permet la description de l'état de la conception pour toutes les étapes du processus. Le fait d'utiliser un langage unique renforce la cohérence entre différentes représentations d'un même système.

• Il renforce aussi la cohérence des outils logiciels utilisés pour la simulation et la synthèse. Il peut être directement compris comme langage d'entrée pour de tels outils.

• Il est indépendant de toute méthodologie de conception, de toute technologie de fabrication et de tout outil logiciel.

• Il supporte plusieurs niveaux d'abstraction et autorise des descriptions hiérarchiques. Il supporte des descriptions comportementales (fonctionnelles) aussi bien que structurelles.

• Il est extensible: il permet au concepteur de définir de nouveaux types d'objets et les nouvelles opérations correspondantes.

• Il est plus qu'un simple format d'échange entre outils logiciels. Il renforce la communication et la cohésion à l'intérieur des équipes de conception et entre les différentes communautés de concepteurs parce qu'il est lisible (format texte) et qu'une description écrite dans un tel langage contient beaucoup d'information sur l'expertise derrière la conception. Il améliore donc grandement les phases de spécification et de documentation.

I.9. Champs d'application du langage VHDL

L'électronicien a toujours utilisé des outils de description pour représenter des structures logiques ou analogiques. Le schéma structurel que l'on utilise depuis si longtemps n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, les fonctions numériques à réaliser ne cessent de prendre de l'ampleur. Il est dès lors

indispensable d'utiliser un langage de haut niveau pour maîtriser la complexité grandissante des systèmes numériques. Le VHDL est l'un des langages modernes et puissants. Il est nettement plus performant que la description par schéma. [6]

Le deuxième point fort du VHDL est d'être un langage de description comportementale de haut niveau. Les anciens langages, ne disposaient pas de cette fonctionnalité. En fait, un langage est dit de haut niveau lorsqu'il fait le plus possible abstraction de l'objet pour lequel il est écrit. Dans le cas du VHDL, il n'est jamais fait référence au composant ou à la structure pour lesquels on l'utilise. Ainsi, il apparaît une notion très importante : la portabilité des descriptions VHDL. Il est également important de noter, que le langage VHDL est normalisé.

Il n'est pas la propriété d'un vendeur d'outils. Cette norme a poussé de grandes sociétés de logiciels à l'utiliser comme langage de description de matériel pour leur outil. Le langage est ainsi devenu un standard reconnu par une majorité des vendeurs d'outils de synthèse.

Le langage VHDL a été initialement utilisé pour la spécification et la simulation de systèmes complexes. Ce langage de haut niveau permet une décomposition hiérarchique et dispose de la notion de temps. Cela a permis de maîtriser la complexité des circuits, particulièrement dans le cas des ASICs. Le même langage est utilisé pour toutes les étapes du développement (spécification, simulation et synthèse). Il permet également la réalisation de bibliothèques de composants réutilisables d'un projet à l'autre.

Ce paragraphe se propose simplement de donner les informations utiles à la compréhension du chapitre qui suit.

I.10. Conclusion

Les simulateurs de la famille Spice proposent de nombreuses primitives (résistances, condensateurs, bobines, transistors,etc…) afin de décrire la topologie d'un circuit, chaque primitive est associée à un modèle plus ou moins précis. Certains modèles, par exemple les transistors MOS sont fortement liés à la technologie en faisant apparaître un nombre important de paramètres de process. D'autre, comme les éléments passifs sont décrits essentiellement à partir de grandeurs électriques. Cependant, quel que soit le type de modèle,

la taille du système à résoudre pour le simulateur sera proportionnelle au nombre de nœuds du circuit [3]. On peut aussi conclure que lorsque le nombre de transistors devient important (quelques centaines à quelques milliers en conception analogique), le temps de simulation devient prohibitif et des problèmes de convergences peuvent apparaître rendant plus complexe la simulation d'un circuit électronique. [16]

Afin de réduire la taille du système matriciel à résoudre, le recours à des modèles de plus haut niveau (amplificateurs opérationnels, comparateurs, etc…) devient alors indispensable.

Chapitre II

LE LANGAGE

VHDL-AMS

II.1. Introduction

Le langage VHDL est un standard IEEE (IEEE 1076-1993) pour la modélisation, la simulation et la synthèse de systèmes matériels (HDL - *Hardware Description Language*). Il est aujourd'hui très largement utilisé et est supporté par tous les environnements d'aide à la conception de circuits et de systèmes électroniques. Le langage VHDL-AMS est aussi un standard IEEE (IEEE 1076.1-1999). Il a été développé comme une extension du langage VHDL pour permettre la modélisation et la simulation de circuits et de systèmes analogiques et mixtes logiques-analogiques. [37]

VHDL-AMS permet de supporter la conception à plusieurs niveaux :

• Modélisation de circuits logiques et analogiques, abstraction possible grâce à des modèles comportementaux de complexités variables (des réseaux de Kirchhoff aux modèles fonctionnels à flot de données).

• Modélisation de systèmes complets (par exemple une chaîne d'acquisition de données d'un capteur avec traitement numérique) avec prise en compte de l'environnement (les effets dûs à la température).

VHDL-AMS offre en outre un support de base pour la modélisation de systèmes non électriques (capteurs, actionneurs, éléments mécaniques). [5]

II.2. Organisation d'un modèle VHDL-AMS

II.2.1. Unités de conception

Un système électronique peut être décrit comme un module ayant des entrées et des sorties [34]. Ce module est appelé dans la terminologie VHDL : **design entity**. Chaque entité peut être décrite comme un ensemble d'unités de conception appelées **design units**. L'unité de conception est le plus petit module peut être compilé séparément. VHDL-AMS offre cinq types d'unités de conception (Figure II.1):

• la déclaration d'entité (*entity declaration*);

• le corps d'architecture (*architecture body*), ou plus simplement architecture;

• la déclaration de configuration (*configuration declaration*);

• la déclaration de paquetage (*package declaration*);

• le corps de paquetage (*package body*). [10]

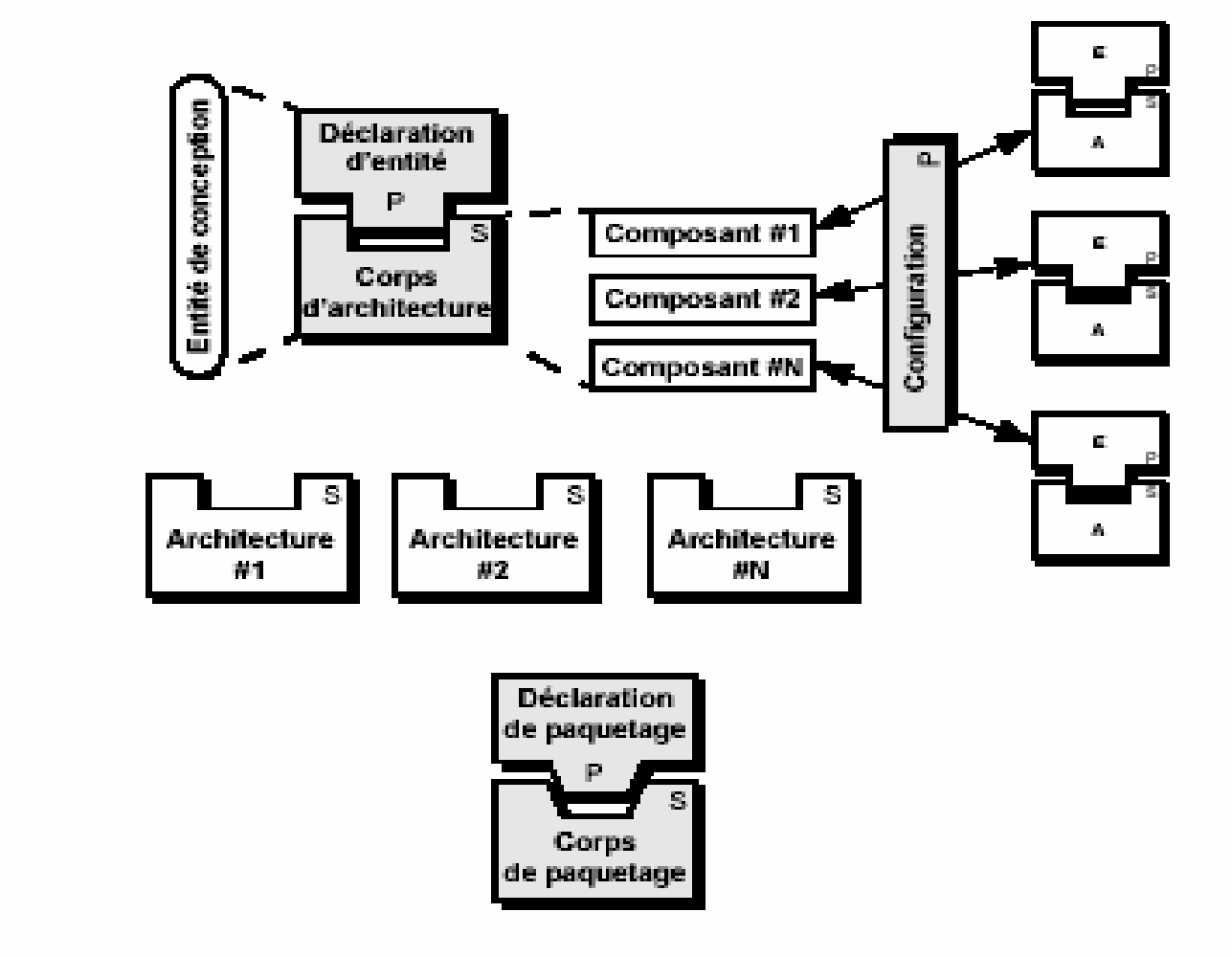


Figure II.1: Unités de conception VHDL-AMS (en gris).

Les trois premières unités de conception (déclaration d'entité, architecture et déclaration de configuration) permettent la description de l'aspect matériel d'un système, alors que les deux dernières (déclaration et corps de paquetage) permettent de grouper des informations pouvant être réutilisées pour la description de plusieurs systèmes différents.

Les trois unités de conception: déclaration d'entité, déclaration de configuration et déclaration de paquetage, sont qualifiées de ***primaires*** (*primary unit*, marquées par un "P" à la Figure 2), car elles décrivent une vue externe (le "quoi" de la boîte noire). Les unités primaires déclarent ce qui est accessible aux autres parties du modèle. Elles permettent le partage et la réutilisation de ressources (modèles et algorithmes). Elles ne contiennent que le minimum d'information nécessaire pour l'utilisation de ces ressources.

Les deux autres unités de conception: architecture et corps de paquetage, sont qualifiées de ***secondaires*** (*secondary units*, marquées par un "S" à la Figure 2), car elles décrivent une vue interne particulière (une réalisation ou le "comment" de la boîte noire). Les unités secondaires regroupent des ressources seulement visibles par leur unité primaire correspondante (déclaration d'entité pour l'architecture et déclaration de paquetage pour le corps de paquetage). Elles contiennent des détails d'implémentation que l'on masque pour :

a) éviter de surcharger l'utilisateur avec des détails inutiles.
b) éviter des modifications malencontreuses de modèles et d'algorithmes de base.

II.2.2. Structure d'un modèle VHDL-AMS [8] [9]

Tout modèle (ou composant) décrit par VHDL-AMS se compose de deux parties (ou objets): la première partie est l'**ENTITY** et la deuxième l'**ARCHITECTURE .**

II.2.2.1. Entité

L'**ENTITY** est la partie (ou l'interface) qui communique entre le monde extérieur et le modèle au moyen de deux objets : les ports (**PORT**) et les paramètres génériques **(GENERIC)** .

Les ports peuvent être de plusieurs classes:

• Les ports de classe ***signal*** (**signal**) définissent des canaux de communication directionnels (entrées (mode **in**), sorties (mode **out**) ou bidirectionnels (mode **inout**)) modélisant des signaux logiques.

• Les ports de classe ***terminal*** (**terminal**) définissent des points de connexions analogiques adirectionnels pour lesquels les lois de conservation de l'énergie (lois de Kirchhoff pour les circuits électriques) sont satisfaits .

• Les ports de classe ***quantité*** (**quantity**) définissent des points de connexions analogiques directionnels d'entrée (mode **in**) et de sortie (mode **out**)) pour lesquels les lois de Kirchhoff ne doivent pas être satisfaits.

Nous pouvons comparer l'**ENTITY** à une boite noire où seuls les nœuds sont visibles, une partie de ces nœuds sont les ports d'entrée/sortie.

Les **GENERICs** sont les constantes (ou des variables statiques), les paramètres, qui peuvent être modifiés par la suite. Pour contrôler nos paramètres d'entrée, nous pouvons ajouter une autre partie qui est optionnelle et qui se trouve entre **BEGIN** et **END** de l'**ENTITY** , ainsi, qu'on peut ajouter des déclarations qui sont de type global.

II.2.2.2. Architecture

L'**ARCHITECTURE** représente une des descriptions possibles de la fonction du modèle. Une **ARCHITECTURE** se réfère toujours à une unique **ENTITY** et contient les déclarations

utilisées dans l'**ENTITY** (les constantes déclarées en **GENERIC** , les nœuds déclarés en **PORT**, etc.).

L'**ARCHITECTURE** contient toutes les déclarations locales, comme par exemple les déclarations des fonctions et des procédures, les constantes, les terminaux, les types, les variables, etc. Elle contient aussi les équations du modèle.

Pour une **ENTITY** donnée, il peut y avoir plusieurs **ARCHITECTUREs** qui lui font appel avec différents types de description et pour une **ARCHITECTURE** donnée, il y a une et une seule **ENTITY** . (figure II.2) .

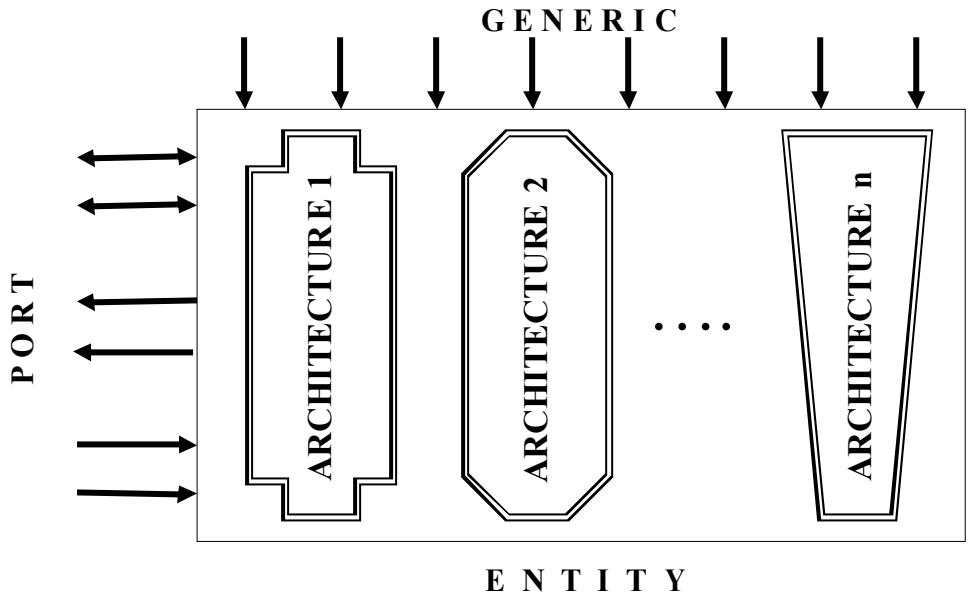


Figure II.2. Structure d'un modèle VHDL-AMS

II.2.2.3.Syntaxe

```
ENTITY  < nom_entité  >  IS
        GENERIC ( < déclaration_GENERIC_1 > ;
                  < déclaration_GENERIC_2 > ;
                   …….
                  < déclaration_GENERIC_N >  );
          PORT  (< déclaration_PORT_1 >;
                   < déclaration_PORT_2 >;
                    ……..
                   < déclaration_PORT_N >  );
END ENTITY  < nom_entité >;
ARCHITECTURE  < nom_arch_1 >  OF  < nom_entité >  IS
                 < déclarations_fonction_procédure >;
                 < déclarations_constantes >;
                 < déclarations_terminaux >;
                 < déclarations_types>;
                 < déclarations_variables >;
BEGIN
                 < type_modèle >;
END ARCHITECTURE  < nom_arch_1 > ;
ARCHITECTURE  < nom_arch_2 >  OF  < nom_entité >  IS
 ……..
ARCHITECTURE  < nom_arch_N >  OF  < nom_entité >  IS
```

II.2.3. Bibliothèques de conception

La compilation d'une unité de conception génère un fichier dont le format n'est pas standard et donc dépend de l'outil VHDL-AMS utilisé. Le résultat de la compilation est stocké dans une ***bibliothèque de conception*** (*design library*) qui est physiquement, p. ex., un répertoire Unix ou un dossier Windows.

Un modèle VHDL-AMS ne considère que des ***bibliothèques logiques*** et l'association à des emplacement physiques doit être faite dans l'environnement VHDL-AMS utilisé.

VHDL-AMS possède deux bibliothèques logiques prédéfinies:

• La bibliothèque de nom logique WORK est le dépositaire de toutes les unités de conception compilées. C'est en fait la seule bibliothèque dans laquelle il est possible d'ajouter et de modifier des éléments.

• La bibliothèque de nom logique STD est prédéfinie et contient deux unités: le paquetage STANDARD, qui inclut les définitions des types, opérateurs et sous-programmes prédéfinis, et le paquetage TEXTIO, qui inclut les définitions et les sous-programmes relatifs au traitement de fichiers textes.

Il est possible de définir et d'utiliser d'autres bibliothèques. La clause **library**, spécifiée avant l'unité de conception, permet de spécifier les noms logiques des bibliothèques utilisées. La figure II.3 donne l'exemple de l'utilisation de la bibliothèque logique IEEE dans laquelle est stocké, entre autres, le paquetage STD_LOGIC_1164 qui définit le type logique à 9 états std_logic et ses opérateurs associés. Il faut noter que dans ce cas le chemin complet d'accès au type est requis. L'utilisation d'une clause **use** permet en plus de relaxer cette contrainte. Le mot-clé **all** rend tous les noms définis dans le paquetage visibles dans le modèle sans qu'il soit nécessaire de les préfixer. Le préfixe peut toutefois être nécessaire si le même nom est défini dans plusieurs bibliothèques déclarées. [9]

```
library ieee ;
entity ...      is
      port (...
         p: in ieee.std_logic_1164.std_logic ;     -- sans clause use
           ) ;
end entity;
```

```
library ieee ;
use  ieee.std_logic_1164.all ;
entity ...    is
      port (...
          p: in std_logic ;       -- avec clause use
             ) ;
end entity ;
```

Figure II.3. Déclaration de bibliothèque pour accès aux éléments du paquetage IEEE STD_LOGIC_1164.

Toute unité de conception possède une ***clause de contexte*** (déclaration de bibliothèque + clause **use**) ***implicite*** (figure II.4).

```
library   std, work ;
use  std.standard.all ;
```

Figure II.4. Clause de contexte implicite à chaque unité de conception.

II.2.4.Terminaux et natures

Le terminal est un objet VHDL-AMS utile pour la spécification de points de connexions pour lesquels les lois de Kirchhoff sont satisfaites. Les terminaux permettent de définir des branches qui elles-mêmes servent de support à la spécification d'équations liant les grandeurs de branches associées, usuellement la tension et le courant pour des systèmes électriques. [20]

Un terminal appartient à une ***nature*** qui représente un domaine d'énergie particulier (électrique, mécanique, thermique, etc.). Chaque domaine d'énergie est caractérisé par deux classes de grandeurs liées à des effets physiques. Les grandeurs "***entre***" (*across*) représentent un effort (p. ex. la tension pour les systèmes électriques, la vitesse pour les systèmes mécaniques, la température pour les systèmes thermiques).

Les grandeurs "***dans***" (*through*) représentent un flot (p. ex. le courant pour les systèmes électriques, la force pour les systèmes mécaniques, la débit de chaleur pour les systèmes thermiques). Ces deux classes de grandeurs permettent de définir la notion d'énergie et de puissance d'un système physique.

Une déclaration de nature en VHDL-AMS ne définit pas directement les grandeurs physiques relatives à chacune des classes "entre" et "dans", mais plutôt leur type, c'est-à-dire l'ensemble des valeurs (nécessairement réelles) que ces grandeurs peuvent prendre. Les grandeurs elles-mêmes seront définies au niveau des branches reliant les terminaux. La figure II.5 donne l'exemple de la déclaration de la nature electrical pour la modélisation de systèmes électriques. [35]

```
Package electrical_systems is
Subtype voltage is real ;       -- dérivé du type real
subtype current is real ;       -- dérivé du type real
-- nature scalaire
Nature electrical is
voltage across       -- type des grandeurs "entre"
current through      -- type des grandeurs "dans"
electrical_ref reference ;       -- terminal de référence
end package electrical_systems ;
```

Figure II.5. Déclaration de la nature electrical.

Cette déclaration définit deux sous-types dérivés du type prédéfini real, appelés **voltage** et **current**. Ces soustypes permettront de caractériser les quantités de branches associées aux terminaux appartenant à la nature **electrical**. La déclaration de la nature **electrical** comprends l'association des (sous-)types à toutes les quantités de branches de classe "dans" et "entre", ainsi que le nom du terminal de référence **electrical_ref**.

Une telle déclaration de nature est normalement incluse dans un paquetage. Il n'existe pas de natures prédéfinies en VHDL-AMS. Dans la suite de ce paragraphe, il sera supposé l'existence d'une bibliothèque logique **ieee_proposed** contenant, entre autres, le paquetage **electrical_systems**. Pour définir des terminaux électriques, il s'agira alors d'inclure la clause de contexte donnée dans la figure II.6. Le contenu des autres paquetages inclus dans cette bibliothèque est décrit dans l'annexe.

```
Library ieee_proposed ;
Use ieee_proposed.electrical_systems.all ;
```

Figure II.6. Clause de contexte pour des entités de conception électriques.

Le but des natures et des types est d'éviter des associations ou des interconnexions qui n'ont pas de sens défini. Par exemple, il est interdit d'associer directement un terminal de nature électrique et un terminal de nature mécanique. Si une telle association est nécessaire, le modèle doit inclure un composant convertisseur (transducteur) approprié.

II.2.5. Objets: constantes, signaux et variables [2]

Les **constantes** ont par définition une valeur fixe qui ne peut être modifiée. La figure II.7 donne quelques exemples de déclarations de constantes. Noter l'usage **d'aggrégats** et l'usage **d'association par nom** pour l'initialisation de constantes de types composites.

```
constant PI: real := 3.1416;
constant ZERO: bit_vector(15 downto 0) := (others => '0');
constant DEF_INST: instruction :=
(op_code => OP_ADD, address_mode => NONE, oper1 => 3, oper2 => 5);
```

Figure II.7: Exemple de déclaration de constantes.

Les **variables** permettent de stocker une valeur d'un type donné et de modifier cette valeur au moyen d'une instruction d'affectation. L'affectation de variable est une instruction séquentielle.

La figure II.8 donne quelques exemples de déclarations de variables.

```
variable count: integer;
variable finished: boolean;
variable current_state: state;
variable register: bit_vector(7 downto 0);
```

Figure II.8 : Exemple de déclaration de variables.

Les **signaux** représentent des formes d'onde logiques sous la forme d'une suite de paires temps/valeur.

A chaque objet de classe signal est associé une structure appelée **pilote** (driver) qui stocke l'historique des valeurs prises par le signal. La valeur d'un signal peut être modifiée par une instruction d'affectation concurrente ou séquentielle . L'affectation peut être faite avec un délai nul ou non nul.

La figure II.9 donne quelques exemples de déclarations de signaux.

```
signal CLK: bit;
signal register: bit_vector(7 downto 0);
```

Figure II.9 : Exemple de déclaration de signaux.

Le choix de l'utilisation d'une variable ou d'un signal dépend de plusieurs facteurs:

• Le domaine d'utilisation de l'objet: une variable a une visibilité plus restreinte qu'un signal car elle n'est visible que dans le contexte d'un processus ou d'un sous-programme. Un signal est global au modèle et constitue l'objet essentiel pour la communication entre processus concurrents et entre composants.

• Le comportement modélisé: un signal ne prend jamais sa nouvelle valeur immédiatement, ce qui peut aboutir à des comportements inappropriés lors de la description d'algorithmes (p.ex. décalages dans le temps d'un cycle d'horloge). Dans ce cas, les variables conviennent mieux.

• Les performances de simulation: une variable n'est qu'une simple position mémoire alors qu'un signal est représenté par une structure de donnée complexe (pilote). De plus, la mise à jour d'un signal est une opération plus coûteuse que la mise à jour d'une variable.

II.2.6. Instructions séquentielles et concurrentes

La base d'un comportement dirigé par les événements est la notion de processus concurrents [2]. Un **processus** (process) en VHDL-AMS définit une portion de code dont les instructions sont exécutées en séquence dans l'ordre donné. Un processus peut posséder sa propre zone de déclarations (constantes, variables, mais pas de signaux, sous-programmes) qui ne sont visibles qu'à l'intérieur du processus.

Les instructions possibles dans un processus sont des instructions de contrôle (condition (**if**), sélection (**case**), boucle (**loop**, **while**, **for**), d'affectation de variables et de signaux, de synchronisation (**wait**) et d'appel de procédure.

La figure II.10 donne un exemple de processus dont l'exécution est activée lorsqu'un événement intervient sur au moins un des signaux de sa **liste de sensibilité** (S1 et S2 ici). Le processus déclare deux variables locales V1 et V2 qui prennent la valeur entière 1 au début de la simulation. Les instructions sont exécutées séquentiellement dans l'ordre donné depuis l'instruction conditionelle **if** jusqu'à l'affectation du signal S2. Le processus est ensuite figé et attend un nouvel événement sur S1 ou S2 pour recommencer son exécution à partir du début, de manière cyclique. Il faut noter que les variables V1 et V2 gardent leurs valeurs d'une exécution du processus à l'autre (à la deuxième activation du processus, on aura V1 = 3 ou 6 et V2 = 2 ou 5, selon la valeur de S1).

```
-- on suppose que les signaux S1 et S2 sont du type bit
P1: process (S1, S2) is -- liste de sensibilité entre parenthèses
variable V1, V2: integer := 1;
begin
if S1 = '1' then
V2 := V1 * 2;
else
V2 := V1 + 4;
end if;
V1 := V2 + 1;
S2 <= not S1 after 5 ns;
end process P1;
```

Figure II.10 : Exemple de processus avec liste de sensibilité.

La synchronisation sur un événement du signal S1 ou S2 peut aussi s'exprimer explicitement à l'aide d'une instruction **wait**. La figure II.11 donne une version équivalente du Code de la figure II.10 illustrant cette possibilité.

```
-- on suppose que les signaux S1 et S2 sont du type bit
P1: process is
variable V1, V2: integer := 1;
begin
if S1 = '1' then
V2 := V1 * 2;
else
V2 := V1 + 4;
end if;
V1 := V2 + 1;
S2 <= not S1 after 5 ns;
wait on S1, S2; -- équivalent à la liste de sensibilité
end process P1;
```

Figure II.11 : Exemple de processus avec instruction wait.

L'instruction **wait** possède d'autres formes permettant la synchronisation sur une condition (**wait until** condition) ou après un certain délai (**wait for** délai).

Il existe d'autres instructions concurrentes en plus du processus dont l'instruction concurrente conditionnelle et l'instruction concurrente sélective. Chacune d'elle possède une formulation équivalente utilisant un processus et une instruction séquentielle conditionnelle ou de sélection.

II.3. Les avantages de VHDL-AMS

Lorsque VHDL-AMS a été créé, il existait de nombreux langages de conception propriétaires pour chaque fondeur ou fournisseur . Ceci était un obstacle à la communication entre domaines scientifiques et posait de graves problèmes aux sociétés lorsqu'un intervenant de la chaîne venait à disparaître ou à être remplacé, car le portage des modèles était alors très délicat et nécessitait de nombreuses heures de travail supplémentaires. [11]

VHDL-AMS est quant à lui un produit non propriétaire et normalisé par l'IEEE qui tend à être reconnu par le plus grand nombre. L'utilisation généralisée de ce langage facilite la communication entre les différents domaines scientifiques grâce à son approche multidomaines native qui permet aussi bien à un électronicien qu'à un mécanicien ou même un chimiste de modéliser la partie d'un dispositif qui le concerne directement sans problèmes de dialogue avec les autres parties.

La normalisation non unilatérale du langage, basée sur une étude de plusieurs années alliant industriels et chercheurs, a permis de généraliser son utilisation et de rendre les projets indépendants des fournisseurs de logiciels choisis, ce qui est primordial pour les entreprises qui étaient extrêmement dépendantes de ce lien auparavant.

La grande force de ce langage est de permettre la simulation mixte en autorisant aussi bien les modélisations à temps continu (analogiques) qu'à évènements discrets (logiques) ou mélangeant les deux. A cette flexibilité d'emploi s'ajoute la possibilité pour les concepteurs d'aborder leurs modèles à différents niveaux d'abstraction. En effet, VHDL-AMS propose des mécanismes permettant de gérer aussi bien les abstractions comportementales (c'est la fonction réalisée par le système qui est modélisée et non sa physique), que les abstractions structurelles (le système est divisé en sous-ensembles qui peuvent eux-mêmes être modélisés au moyen de différentes abstractions, …) ou bien de type work-flow (enchaînement de blocs fonctionnels dont les entrées n'ont pas d'influence sur les sorties des blocs précédents). Les modèles créés avec VHDL-AMS peuvent donc aussi bien être descriptifs que prédictifs.

II.4.Les limites de VHDL-AMS

VHDL-AMS n'est cependant pas à même de proposer des instructions répondant à tous les besoins des concepteurs de modèles. Par exemple, l'utilisation des dérivations spatiales n'est pas prévue par le langage, ce qui rend délicat les modélisations géométriques. Seules les dérivations temporelles sont acceptées par VHDL-AMS. Par ailleurs, même si le langage est à même de supporter des palliatifs à ses manques grâce à ses possibilités d'interfaçage avec d'autres langages (notamment le C/C++), la forme de ces interfaces n'est pas standardisée. De ce fait, les modèles ayant recours à des langages extérieurs à VHDL-AMS ne sont généralement pas portables. Même si VHDL-AMS laisse à l'utilisateur la possibilité de définir ses propres natures, il n'offre pas d'alternative possible à la sémantique de connexion faisant intervenir les lois de Kirchhoff généralisées. Cela devient un handicap lorsque l'on veut traiter d'autres systèmes de relations physiques. Il n'est, par exemple, pas possible de traiter la propagation des ondes électromagnétiques au moyen des terminaux, car les règles associées à cette propagation ne vérifient pas les lois de Kirchhoff. [3] [31]

Enfin, le fait que les simulateurs actuels soient basés sur des extensions et des modifications d'anciens simulateurs, et pas encore sur de nouvelles techniques de simulation spécifiques, implique des limitations dans les possibilités de simulation qui empêchent l'implémentation de certaines instructions du langage.

II.5. Les évolutions liées à VHDL-AMS

Avec la réaffirmation de la normalisation VHDL-AMS prévue, et son évolution future, une partie des problèmes présentés ci-dessus devrait trouver une solution. Par ailleurs, le langage verra ses possibilités prochainement étendues grâce à l'extension Radio Fréquence/Micro Ondes (Radio Frequency/ Micro Waves - RF/MW) en cours de développement. Des pistes complémentaires sont également étudiées pour élever le niveau d'abstraction des connexions,

afin d'autoriser l'utilisation de différentes classes de signaux sur un même dispositif de connexion en fonction du niveau d'abstraction du modèle. Il est également question d'incorporer des primitives SPICE à VHDL-AMS et d'y adjoindre des mécanismes automatiques de spécification et de vérification des modèles. Par ailleurs, avec les possibilités grandissantes offertes par la nouvelle génération de simulateurs VHDL-AMS, il est légitime de se demander si les efforts de recherche pour développer des simulateurs spécialement dédiés à la simulation analogique supportant le jeu complet d'instructions VHDL-AMS seront encouragés. [11] [32]

II.6. Conclusion

Il serait fastidieux de rappeler dans ce chapitre les bases du langage VHDL-AMS. Le lecteur pourra se reporter le cas échéant aux nombreuses publications parues sur le sujet, entre autres les travaux de Yannick Hervé et Alain Vachoux.

Cependant il semble intéressant d'insister sur certains avantages liés à l'utilisation de ce langage :

- la compatibilité ascendante avec VHDL permet de simuler des blocs numériques, y compris synthétisables, avec les blocs analogiques. Ceci autorise la mise au point d'une partie numérique intégrée en fonction des performances imposées de la partie analogique (cellules de bibliothèque par exemple), ou réciproquement déterminer les performances de la partie analogique et mixte (résolution des convertisseurs ou fréquence d'échantillonnage par exemple) pour un fonctionnement correct d'algorithmes numériques.
- la modélisation comportementale permet la fourniture de modèles analogiques et mixtes en respectant les contraintes de propriété intellectuelle; pour l'instant, la synthèse analogique n'étant pas encore opérationnelle, on peut se contenter de modèles de très haut niveau.
- VHDL-AMS est un langage multi-technologies capable de décrire des systèmes tant électroniques que mécaniques, thermiques ...

Chapitre III

MODELISATION COMPORTEMENTALE VHDL-AMS

III.1. Introduction

Le temps de calcul des circuits électroniques devient très important, même prohibitif lorsque le nombre de transistors est très important (au-delà de 1000 transistors) , la nature du circuit étant également un facteur déterminant (fonction non linéaire occasionnant des problèmes de convergence), la solution consiste alors à élaborer un modèle à un niveau plus élevé que celui du transistor. La modélisation comportementale constitue une de ces solutions : au lieu de décrire un circuit transistor par transistor (niveau structurel), on tente de modéliser son comportement électrique externe (niveau comportemental). [16]

Une démarche plus proche de la philosophie VHDL-AMS consiste à modéliser uniquement le comportement d'un système. Il est ici tenu compte des lois de conservation de l'énergie et donc des interactions pouvant s'exercer entre les entrées d'un modèle comportemental et les sorties de son prédécesseur. [9]

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

III.2. Principe de la modélisation comportementale

III.2.1. Les classes de représentation

Tout système physique, quel qu'il soit, peut être représenté de deux manières différentes : [3] [8]

1. Par une description comportementale : indépendante de l'architecture du système, elle en explicite le fonctionnement par l'écriture de modèles, c'est-à-dire de représentations mathématiques associant des variables et des équations matérialisant les relations entre ces variables .

2. Par une description structurelle: elle matérialise directement l'architecture interne du système par l'association d'éléments qui peuvent eux-mêmes être décrits au niveau inférieur

D'une manière générale, pour les systèmes électroniques, le passage automatique de la description comportementale à la description structurelle est appelé synthèse. Si cette opération est déjà maîtrisée depuis longtemps en ce qui concerne les circuits numériques, elle est tout à fait d'actualité pour les circuits analogiques.

III.2.2. Différents niveaux d'abstraction

Pour chacune des deux représentations (comportementale et structurelle), il est possible de définir quatre niveaux d'abstraction, que ce soit pour le numérique ou l'analogique, résumés dans les tableaux **III.1 et III.2**. [3]

Niveau d'abstraction	Comportemental	Structurel
Système	Synoptiques, Algorithmes	Processeurs, mémoires
Micro-architecture	Register Transfert Level (RTL)	Registres, ALU
Logique	Equations booléennes Diagrammes d'état	Portes logiques
Circuit	Fonctions de transfert Diagrammes temporels	Transistors

Tableau III.1. Niveaux d'abstraction en numérique.

Niveau d'abstraction	Comportemental	Structurel
Système	Fonctions de transferts Diagrammes H(s),H(z)	Convertisseurs, PLL, filtres, intégrateurs, multiplieurs…..
Fonctionnel	Equations algébriques linéaires et non linéaires, Tables	Amplificateurs opérationnels , comparateurs, sources……
Circuit	Macro modèles	Transistors, R,L,C
Composant	Modèles de composants	Layout des composants

Tableau III.2. Niveaux d'abstraction en analogique.

Le sens de passage d'un niveau d'abstraction à un autre définit le type de méthodologie utilisée :

III.2.2.1. La méthodologie descendante " Top-Down"

Est basée sur une suite de raffinements successifs partant d'un cahier des charges pour aboutir à une description détaillée de la réalisation. Le cahier des charges définit le "quoi", c'est-à-dire principalement les fonctions à réaliser et les conditions dans lesquelles ces fonctions devront s'exécuter. A l'autre bout du processus, la réalisation décrit le "comment", c'est -à- dire la manière qui a été retenue pour fabriquer un circuit satisfaisant les contraintes imposées par le cahier des charges [4]. La méthodologie descendante permet de vérifier le bon fonctionnement du système avant de passer à une description niveau transistor et de détecter des erreurs de conception précoces. Elle permet également de reporter le choix de la technologie le plus tard possible dans le cycle de conception. Ainsi, une modification de la technologie ne remet pas en cause les premières étapes de la conception. La figure **1** détaille les différentes étapes suivies dans une approche Top-Down. [2]

III.2.2.2. La méthodologie montante "Bottom-Up"

Est surtout utilisée en analogique. À partir d'un schéma établi au niveau transistor, une caractérisation est effectuée et utilisée pour paramétrer un modèle comportemental de plus haut niveau de ce schéma [3], comme illustré par la figure III.1, ce modèle est ensuite utilisable pour une simulation globale du circuit considéré . La méthodologie montante est bien adaptée à la réalisation de circuits dont la structure est essentielle à leur bon fonctionnement. [4]

Dans le flux de conception Top-Down et Bottom-Up, on peut mélanger dans une même description plusieurs niveaux d'abstraction. Ainsi, on peut simuler un système avec des blocs décrits au niveau fonctionnel et d'autres décrits au niveau transistor. C'est la technique de multi-abstraction.

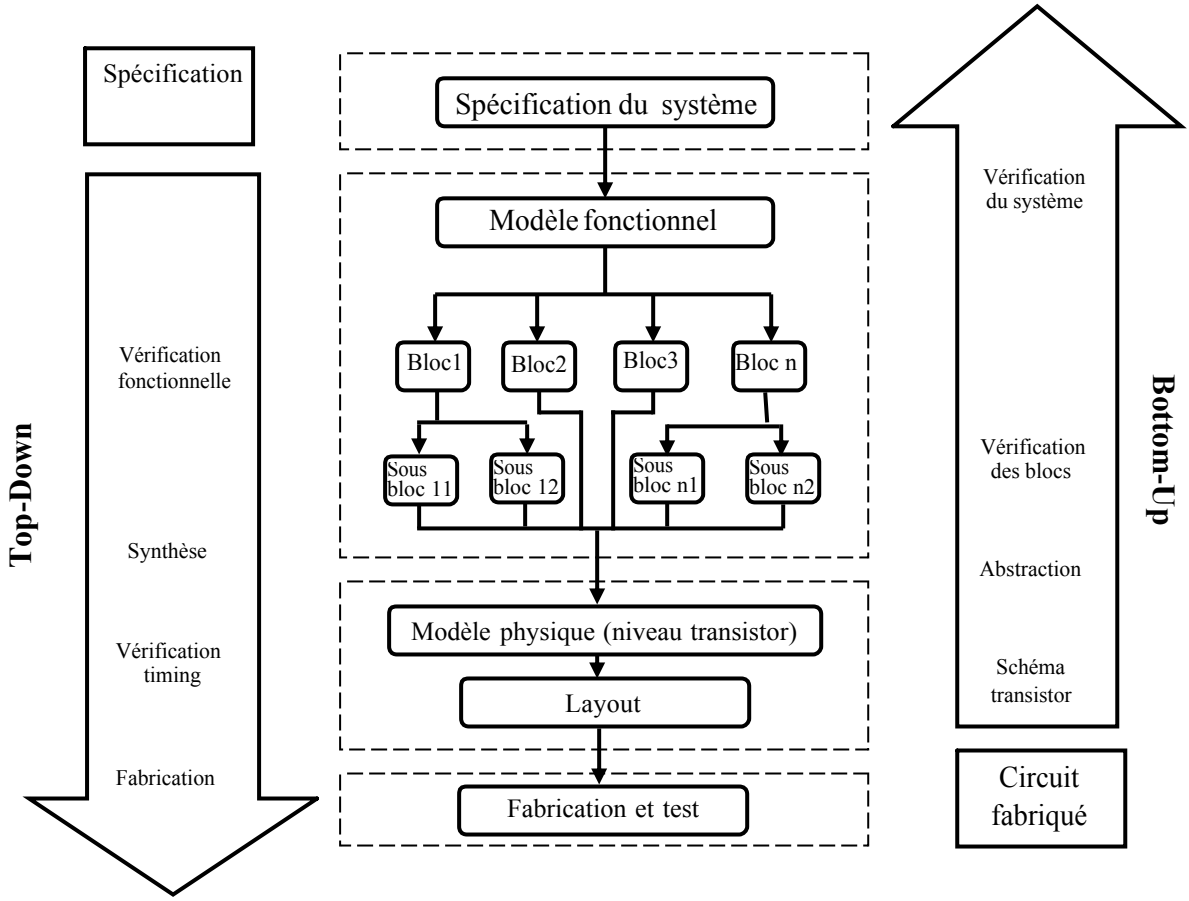


Figure III.1 : Méthodologie de conception Top-Down et Bottom-Up.

III.3. Méthodologie de modélisation

III.3.1. Présentation et objectifs

Le modèle d'un système est une représentation de son comportement à l'aide de laquelle le simulateur comprend et procède à des calculs. Un modèle doit être le plus fidèle et le plus exact possible, c'est le plus important critère de la modélisation. Mais écrire un modèle exact est la tache la plus difficile. [8] [10]

Il y a différentes façons de modéliser le comportement d'un système. Le modèle peut être à temps discret ou à temps continu ou les deux en même temps, et ce comportement doit être compréhensible par le simulateur.

Deux façons permettent de représenter un modèle à temps discret:

- Par la communication des processus (signal).
- Par les équations booléennes.

Or, un modèle à temps continu ne peut être représenté que d'une seule façon:

- Par des équations analytiques ou mathématiques.

Comme il est mentionné ci-dessus, l'élément indispensable dans la simulation est le modèle. Un des objectifs de la simulation est de vérifier la correcte fonctionnalité du système. Dans le domaine électrique, la simulation permet la validation d'un circuit électrique quelle que soit sa nature numérique ou analogique ou les deux en même temps. Pour un circuit numérique nous utilisons la logique discrète (0, 1, X, etc.), pour un circuit analogique nous utilisons la propriété de temps continu pour décrire le comportement du circuit en utilisant des composantes de base (résistance, capacité, etc.). les progrès accomplis en VLSI, ont permis de combiner les deux sous-systèmes analogique et numérique,. Cette évaluation a mené à créer des nouveaux types de simulateurs, qui sont appelés des simulateurs mixtes.

III.3.2. Types de formalismes

III.3.2.1. Réseaux de Kirchhoff

La théorie des réseaux de Kirchhoff permet de définir des modèles de circuits électriques, et même de systèmes non électriques par analogie entre grandeurs physiques, comme des modèles mathématiques obéissant aux règles suivantes:

• Un réseau de Kirchhoff est une connexion d'un nombre fini d'éléments.

• Un élément possède un certain nombre de bornes (terminaux).

• A chaque borne d'un élément sont associées deux fonctions à valeurs réelles d'une variable réelle, le temps: le potentiel $v(t)$ de la borne et le courant $i(t)$ pénétrant dans la borne.

• Un ensemble de bornes connectées forme un noeud. Les bornes connectées au même noeud ont le même potentiel. La somme des courants pénétrant par les bornes connectées au même noeud est nulle.

• Un élément est caractérisé par un certain nombre de relations constitutives indépendantes entre les potentiels et les courants à ses bornes. La somme des courants pénétrant par les bornes de l'élément est nulle. Les relations constitutives ne font intervenir que les différences de potentiel, ou tensions, entre les bornes. Les relations constitutives peuvent être linéaires ou non linéaires.

• Un réseau de Kirchhoff n'a pas de dimension; on suppose qu'il y a propagation instantanée des phénomènes. Ce formalisme n'est donc plus valable si le circuit fonctionne à hautes fréquences.

• Les contraintes topologiques imposées aux noeuds d'un réseau de Kirchhoff imposent ***la conservation de l'énergie*** dans le réseau: la somme des puissances dissipées dans les branches du réseau est nulle.

La Figure III.2 illustre la représentation d'un circuit amplificateur MOS au moyen d'un réseau de Kirchhoff. Les éléments complexes (transistors, diodes) sont eux-mêmes modélisés comme des circuits équivalents formés d'éléments simples (résistances, capacités, sources). [10]

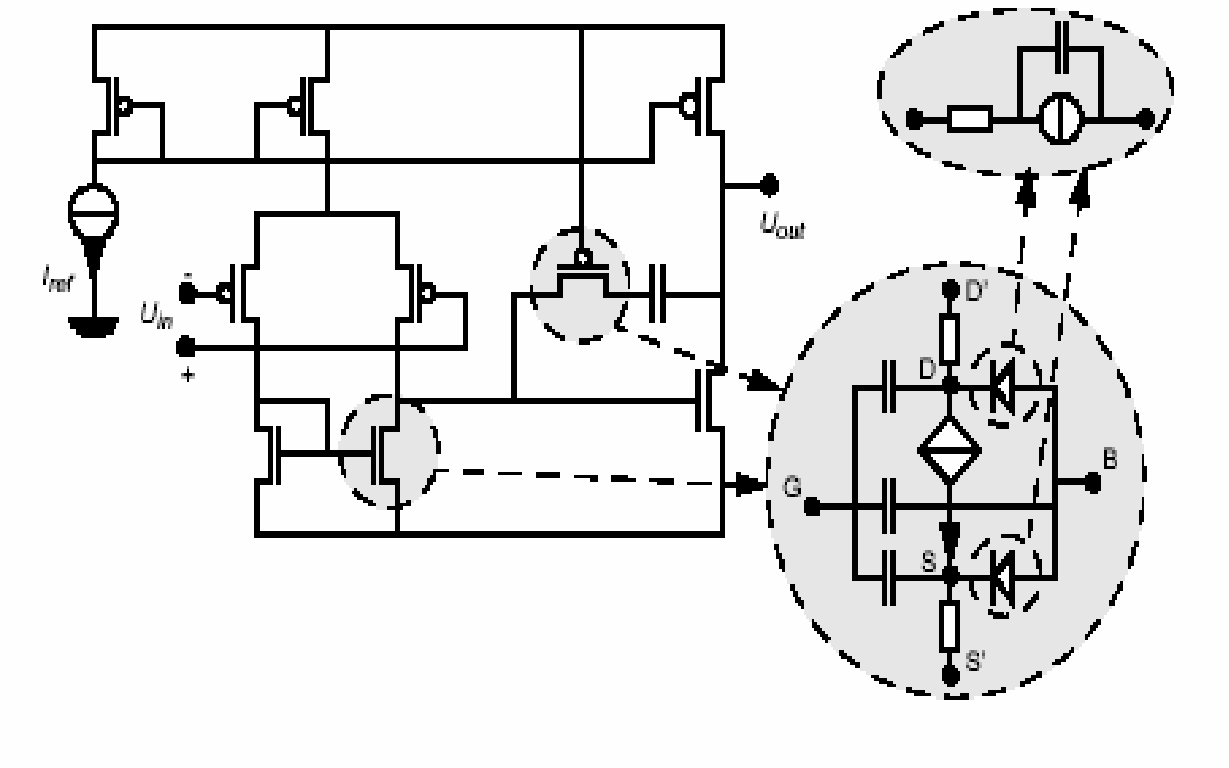


Figure III.2. Amplificateur CMOS représenté par un réseau de Kirchhoff.

III.3.2.2. Diagrammes de blocs

Les diagrammes de blocs offrent un formalisme plus abstrait pour lequel les lois de conservation de l'énergie ne sont plus considérées. Un diagramme de blocs est une connexion d'un nombre fini de blocs, chaque bloc possédant des ports d'entrée et de sortie. Un bloc est caractérisé par un ensemble de relations entrées-sorties indépendantes, par ex.: sommation, fonction de transfert, intégration. Les relations entrées-sorties peuvent être linéaires ou non linéaires. Les blocs sont interconnectés par des chemins orientés représentant des signaux qui peuvent être des fonctions à valeur réelles d'une variable réelle, usuellement le temps (système continu), des fonctions à valeur réelles d'une variable entière, usuellement le temps (système discret) ou des fonctions à valeurs discrètes d'une variable entière, usuellement le temps (système logique).

III.3.2.3. Processus concurrents

Le formalisme des processus concurrents permet de représenter le comportement d'un système dont le comportement est dirigé par les événements (*event-driven behaviour*). Ce formalisme est approprié pour modéliser le comportement de systèmes matériels logiques (*digital hardware systems*).

Un processus définit une séquence d'opérations qui sont exécutées en fonctions d'événements auxquels il est sensible (*trigger conditions*). Les opérations sont usuellement des opérations logiques ou arithmétiques avec un contrôle du flot d'exécution (condition, boucle). Dès qu'un processus est activé il exécute ses instructions jusqu'à un point d'arrêt. Le point d'arrêt est une instruction particulière qui dit essentiellement d'attendre jusqu'à ce qu'il y ait un nouvel événement sur les signaux sensibles du processus. L'exécution d'un processus est cyclique: la séquence d'instructions recommence au début une fois la dernière instruction exécutée. Chaque processus peut être activé de manière concurrente et asynchrone. Un "super processus" (séquenceur, scheduler) est défini pour contrôler l'activation des processus.

Un réseau de processus est interconnecté par des signaux qui sont des fonctions à valeurs discrètes d'une variable entière, usuellement le temps (système logique).

III.3.2.4. Macromodèle

Un macromodèle est une représentation simplifiée, mais précise, d'un circuit ou d'un système à un niveau d'abstraction donné. Un macromodèle est très souvent défini au niveau électrique, ou circuit. Un macromodèle est usuellement basé sur la topologie ou la structure du circuit original, mais comporte beaucoup moins de composants car il modélise principalement les caractéristiques vues de l'extérieur du circuit. Le résultat est qu'un macromodèle est simulé plus rapidement que le modèle original (généralement par un facteur 5 à 10). Il est aussi possible de définir un macromodèle comportemental sous la forme d'un ensemble d'équations simplifiées.

III.3.2.4.1. méthodologie de la macro-modélisation

Pour décrire de façon structurelle, SPICE a défini un langage pour la modélisation des fonctions analogiques appelé **macro-modélisation**. Ce type de modélisation est utilisé comme une modélisation comportementale. Les macro-modèles sont implantables sur n'importe quel outil de simulation à base de SPICE.

Nous savons tous que le réseau ou circuit électrique décrit en netliste de type SPICE est analysé par le simulateur lui-même pour construire un système d'équations basées sue les lois de Kirchhoff, donc, basées sur une loi de conservation d'énergie et sur les équations des composants. La macro-modélisation consiste à remplacer une partie d'un circuit ou d'un système par un autre modèle structurel plus simple (nombre de nœuds réduit) et plus prés du circuit initial. Dans un autre sens, la macro-modélisation consiste à satisfaire des spécifications externes sans regarder la topologie initiale du circuit. Le choix de l'un ou de l'autre, dépend du niveau d'abstraction. Le but essentiel de la macro-modélisation est de réduire la taille du circuit et ainsi de réduire le temps de simulation.

Les macro-modèles sont construits à partir d'un nombre réduit de composants idéaux. Les composants utilisés sont des composants primitifs de SPICE. Nous pouvons inclure des éléments passifs (résistance, capacité, etc.), des sources dépendantes et indépendantes, et des modèles non-linéaires de bas niveau comme les diodes et les transistors bipolaires ou MOS de niveau 1 de SPICE. [10]

Les sources contrôlées (sources dépendantes) sont des éléments idéaux qui permettent d'exprimer des relations mathématiques entre les courants et les tensions. SPICE définit quatre types de sources contrôlées:

• 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 :

$i = f\ (\ i_1, \ldots\ i_n)$.

Les éléments passifs (capacité, inductance, etc.) sont utilisés pour réaliser des opérations de dérivation et d'intégration. Ces opérations de base sont utilisées pour décrire des fonctions de transfert en "s". [9]

La méthode de macro-modélisation consiste à utiliser trois blocs de base composés par la figure III.3 :

1. Un étage d'entrée pour implanter l'impédance d'entrée.

2. Un étage correspondant à la fonction principale (fonction de transfert du circuit).
3. Un étage de sortie pour implanter l'impédance de sortie.

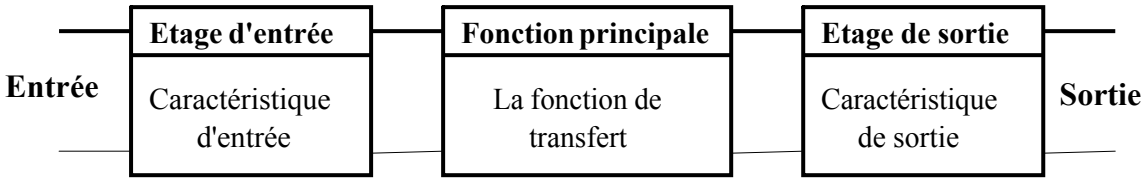


Figure III.3. Méthodologie de base de la macro-modélisation.

III.4. Modélisation comportementale VHDL-AMS

III.4.1. Modélisation comportementale analogique [3] [8]

Les aspects continus des comportements des parties de système visés par la norme VHDL 1076.1 peuvent être décrits par un système d'équations différentielles et algébriques (DAE) avec le temps comme variable indépendante. Ces équations sont de la forme : (équation III.1)

$$F\left[x, \frac{dx}{dt}, t\right] = 0 \qquad \text{(III.1)}$$

Avec x le vecteur des inconnues, dx/dt le vecteur des dérivées des inconnues et t le temps. La plupart de ces systèmes d'équations n'ont pas de pure solution analytique, donc en pratique les solutions doivent être approximées en utilisant des techniques numériques.

Les modèles de composants interconnectés de VHDL-AMS forment un système gouverné par des équations différentielles et algébriques dont la solution est délivrée selon le temps par un solveur analogique et le résultat est affecté aux inconnues déclarées dans le modèle qui seront accessibles périodiquement pour l'affichage des courbes.

Remarquons que le terme « électrique » n'est pas utilisé. Les ensembles d'équations différentielles et algébriques décrites par le système analogique sont des équations mathématiques et le solveur ne fait pas la différence entre électrique et non électrique. C'est pour cela que l'on utilise le terme «multi-technologique ».

Il est nécessaire de faire la différence entre deux types de DAE (figure III.4) :

- Non-conservatif.
- Conservatif

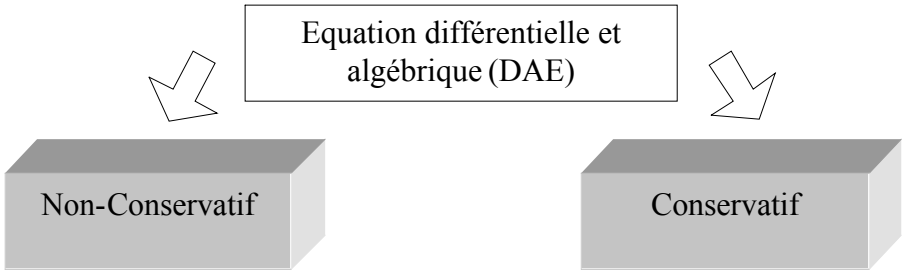


Figure III.4. Types des équations différentielles et algébriques.

III.4.1.1. Les systèmes analogiques conservatifs

Dans un système conservatif, les équations différentielles et algébriques contiennent des inconnues définies explicitement et sont régies par les lois de la physique et les lois de conservation d'énergie. Les circuits électriques sont de bons exemples de systèmes à temps continu et sont gouvernés par des ensembles d'équations conservatives (lois de Kirchhoff). De même les systèmes mécaniques sont gouvernés par des équations conservatives obéissant aux lois de Newton et aux lois de la cinétique. Les structures et les formalismes des systèmes conservatifs sont exploités dans VHDL-AMS par des instructions telles que **terminal** et **nature** qui ont été décrites dans le deuxième chapitre. Pour cela, également on cite deux exemples d'équations d'un système conservatif : (équation III.2 et III.3)

$$Vitesse = \frac{d(déplassement)}{dt} \quad \text{et } V=R\ I \qquad \text{(III.2) et (III.3)}$$

III.4.1.2. Les systèmes analogiques non-conservatifs

Le comportement dynamique du système non-conservatif est décrit à l'aide d'une DAE, mais il n'est pas nécessaire qu'il soit à conservation d'énergie. Cette terminologie n'implique pas les équations différentielles et algébriques, car les inconnues ne sont pas définies et les équations ne sont pas produites en utilisant les constructions de VHDL-AMS qui fournissent

explicitement et systématiquement identifiant tout en utilisant les lois de physique et la conservation d'énergie.

Pour illustrer les systèmes analogiques non-conservatifs et pour montrer la capacité de VHDL-AMS à décrire les équations mathématiques, on peut citer un exemple d'équation différentielle du premier ordre d'un système non-conservatif : :(équation III.4)

$$\frac{dU(t)}{dt} = C_0 U(t) \qquad \text{(III.4)}$$

III.4.2. Représentation des grandeurs analogiques

III.4.2.1. Bornes (terminals)

VHDL-AMS, comme nous l'avons dit, permet de décrire des systèmes conservatifs. Dans ce cas, le langage définit des bornes ou TERMINALs, qui sont en fait les nœuds analogiques pour lesquels les lois de conservation de l'énergie (lois de Kirchhoff pour l'électricité) doivent s'appliquer. Ces bornes peuvent soit être internes à un circuit (et donc définies à l'intérieur d'une architecture), soit servir d'accès analogiques externes (dans ce cas, elles sont définies dans la partie PORT de l'entité comme le montre l'exemple suivant : (figure III.5): [10] [37]

```
Library ieee_proposed;
use ieee_proposed.electrical_systems.all ;
entity inductance is
generic (L: real) ;
port (terminal t1, t2: electrical ) ;
end entity inductance ;
architecture bhv of inductance is
quantity UL across IL through t1 to t2 ;
begin
    UL == L * IL'dot ;
end architecture bhv ;
```

Figure III.5. Modèle VHDL-AMS d'une inductance.

Ces bornes ont pour type une nature relative à une discipline de la physique. Ces natures sont définies dans des packages spécifiques dont s'occupe le Working groupe IEEE1076.1.

On peut citer par exemple le package **ELECTRICAL_SYSTEMS** qui définit entre autres la nature **electrical.**

III.4.2.2. Quantités

VHDL-AMS permet de représenter les inconnues des équations différentielles (DAEs) sous forme de **QUANTITY.** Ces quantités peuvent être soit de type standard VHDL (réel par exemple), soit plus spécifiquement d'un sous-type relatif à une nature définie précédemment (exemple : **voltage** ou **current** pour la nature **electrical**).

Ces quantités peuvent être soit libres (intermédiaires de calcul ou bien interfaces pour des systèmes non-conservatifs), ou bien liées à des bornes. Dans ce dernier cas, pour un couple de bornes donné, on peut définir des quantités **ACROSS** et des quantités **THROUGH**. Une quantité **ACROSS** est homologue à un effort (tension en électricité), et une quantité **THROUGH** à un flux (courants), plusieurs quantités **THROUGH** entre deux mêmes bornes sont automatiquement additionnées par le simulateur. [37]

Il est nécessaire en VHDL-AMS de bien distinguer quantités et signaux. Ces derniers peuvent être eux aussi réels et non plus seulement énumérés comme en VHDL (bit, std_logic…), cependant, alors que les quantités sont définies en temps continu, les signaux le sont toujours en temps discret et prennent donc leurs valeurs à des instants précis.

III.4.3. Equations différentielles

Les relations entre les quantités sont exprimées par des équations différentielles algébriques par rapport au temps. Afin de les différencier des affectations de signaux et variables ainsi que des tests, elles utilise la syntaxe == .

Le sens d'écriture de ces équations n'a aucune importance pourvu qu'elles comprennent au moins une quantité (elles peuvent aussi contenir des variables, signaux…), le simulateur tente simplement d'égaliser numériquement les deux membres de l'équation. [8]

Une règle capitale en VHDL-AMS pour la résolubilité des modèles est que le nombre d'équations doit être égal à la somme du nombre de quantités **THROUGH** et du nombre de quantités libres. Si ce n'est pas le cas, ceci signifie en général qu'il y a un problème d'écriture dans le modèle.

III.4.4. Dérivation / Intégration

La dérivation et l'intégration d'une quantité par rapport au temps s'effectuent respectivement par les attributs **'dot** et **'integ** . L'exemple de l'inductance permet d'illustrer l'utilisation du premier cité.

Ces attributs sont cascadables : il est possible d'obtenir la dérivée seconde par **'dot 'dot** . bien souvent, il est préférable d'utiliser l'attribut **'dot** , car **'integ** est beaucoup plus sensible aux conditions initiales, qui doivent être donc être spécifiées de façon très explicite.
On peut également ranger sensiblement dans cette catégorie l'attribut **'delayed** qui, comme pour les signaux VHDL, permet d'effectuer un décalage temporel d'une quantité par rapport à une autre.

III.4.5. fonction de transfert

VHDL-AMS offre la possibilité d'assigner directement à une quantité le résultat du produit d'une autre par une fonction de transfert. Cette dernière peut être définie soit par transformée de Laplace (attribut **'ltf**) pour les systèmes continus, soit par sa transformée en Z (attribut **'ztf**) pour les systèmes échantillonnés. Dans les deux cas, les pôles et les zéros de la fonction sont donnés par des vecteurs de constantes réelles.

III.4.6. détection de seuil

L'attribut **'above** , appliqué à une quantité, vaut **true** si la quantité est supérieur au seuil spécifié, **false** sinon. Cet attribut a surtout pour avantage de générer un événement vers le simulateur numérique, ce qui permet de l'utiliser pour synchroniser des processus VHDL.

Réciproquement au cas précédent, cette construction permet donc de transcrire des grandeurs analogiques dans le monde numérique, par exemple pour un trigger d'entrée logique :

```
begin
if  Vin'above (Vih)  then
Input  <=  '1';
elsif  Vin'above (Vil)  then
Input  <=  'x';
else
Input  <=  '0';
end if ;
```

III.4.7. Alternatives

VHDL-AMS rajoute entre autres à VHDL la construction **IF … USE … ELSE … END USE.** Elle permet d'effectuer un choix entre plusieurs DAEs selon une condition. Ceci permet par exemple de modéliser un limiteur :

```
if  Vin'above (Q)  use
Vout == Q ;
else
Vout == Vin ;
end use ;
```

III.4.8. Discontinuités

Le simulateur VHDL-AMS doit être capable de gérer correctement dans le domaine temporel les discontinuités des quantités ou de leurs dérivées, mais l'algorithme utilisé ne les détecte pas automatiquement. Si une telle discontinuité se produit dans ces conditions, le pas du simulateur n'étant pas modifié, la convergence de l'algorithme de résolution des équations différentielles est plus tardive, et celui-ci produit donc des résultats incorrects juste après cette discontinuité.

La solution consiste donc à réduire le pas de simulation aux alentours des discontinuités, l'algorithme effectuant si nécessaire un retour en arrière. Ceci est réalisé par la construction **Break on** suivi de la quantité subissant la discontinuité. Cette construction peut être également utilisée avec un signal afin de réinitialiser le simulateur lors du changement d'état de celui-ci. [8]

III.5. Conclusion

L'utilité de la modélisation comportementale VHDL-AMS pour la conception des circuits électronique 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: [3]

• 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 a permis de mettre en valeur les buts de la modélisation comportementale VHDL-AMS:

- 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 (top-down) associée à une validation (bottom-up).

Chapitre IV

IMPLEMENTATION

VHDL-AMS

IV.1. Introduction

Dans notre époque informatique bercée par des environnements graphiques gérés à l'aide de la souris, quoi de plus naturel et intuitif pour les utilisateurs non spécialistes des langages que l'assemblage d'icônes représentant les fonctions des systèmes. [28]

Cette interface graphique utilisateur (Graphical User Interface – GUI) est un des aspects développés par la nouvelle génération de simulateurs. Elle permet de cacher un code HDL derrière l'image de sa fonction, comme l'illustre la figure IV.1. De cette manière, l'utilisateur non spécialiste se réfère dans un premier temps à la fonction qu'il veut réaliser et non à la manière dont elle est implémentée.

```
-- ------------------------------------------------------
-- Copyright (c) 2002 by ANSOFT Corp. All rights reserved. --
-- ------------------------------------------------------
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY R IS
GENERIC (R : REAL := 1.0e+3);
PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY;
ARCHITECTURE behav OF R IS
QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
v == i*R;
END behav;
```

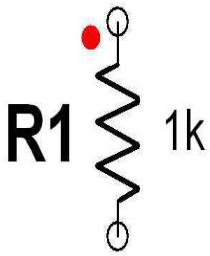


Figure IV.1 : Code VHDL-AMS et représentation graphique associée d'une résistance.

Pour ce faire, l'interface graphique n'a besoin de connaître que les informations contenues dans la déclaration des modèles (ENTITY) qui spécifient les entrées/sorties et les paramètres.

L'implémentation des modèles (ARCHITECTURE) n'intervient pas dans cette représentation, mais permet ensuite à l'utilisateur de choisir un niveau de modélisation parmi ceux éventuellement prévus par le concepteur du modèle. Par des interfaces relativement simples et conviviales, l'utilisateur peut ainsi construire un modèle en "instanciant" graphiquement ses composants, en les connectant entre eux et en traitant le résultats souhaités. En ce qui concerne les connexions, qui peuvent être de différentes classes (chaque

classe transportant plusieurs types d'informations) dans VHDL-AMS, leur intégrité est vérifiée par l'environnement de simulation au moment où elles sont dessinées par l'utilisateur, et une couleur peut leur être attribuée en fonction du domaine de la physique auquel elles font référence.

En effet, l'outil graphique, qui est une abstraction du code VHDL-AMS, représente une limitation pour les développeurs comme toute abstraction. A titre d'exemple, il est effectivement tout à fait impossible de rendre compte précisément de tous les comportements possibles d'un système à l'aide d'un modèle comportemental. Il aura cependant l'intérêt d'être rapide, le plus souvent simple comparé au modèle physique, il ne nécessitera pas de connaissances spécifiques dans un domaine particulier de la physique et sera facilement intégrable dans un système plus important. Sa fonction est de rendre compte du fonctionnement "normal" du dispositif. Il en va de même pour le GUI du logiciel qui représente une simplicité d'utilisation, mais fait perdre une partie de la souplesse et des possibilités du langage. Ceci rend impossible une conception entièrement graphique des composants élémentaires. Il est donc indispensable pour les développeurs de modèles de continuer à bénéficier d'une interface texte leur permettant d'utiliser l'intégralité des possibilités offertes par les langages afin d'offrir les modèles les plus complets et les plus polyvalents possibles aux utilisateurs non spécialistes.

Les nouveaux environnements de travail pour les HDLs doivent donc inclure en leur sein une représentation symbolique simple des composants et de leurs connectiques, tout en conservant la conception de modèles à l'aide des outils textes classiques.

IV.2. Le choix du logiciel de simulation

Comme énoncé précédemment, notre choix s'est porté sur VHDL-AMS, en partie à cause de la diversité des simulateurs disponibles pour ce langage. En effet, l'offre logicielle dans ce domaine est très importante, mais également très variée. Outre la concurrence qui fait naître des outils similaires, l'évolution dans l'utilisation du langage lui-même a fait émerger plusieurs philosophies de conception. Nous avons donc dû choisir parmi tout cela le simulateur le plus adapté à nos besoins.

IV.2.1. Les simulateurs de première génération à code apparent

Ces outils (hAMSter, ADVance MS, SMASH, …) appartiennent pour la plupart à une génération de simulateurs qui tend à devenir obsolète [15]. En effet, ils ne s'inscrivent pas dans le cadre de l'accessibilité aux non spécialistes du langage. Il n'en reste pas moins que ce sont des outils de conception très puissants et intéressants pour les spécialistes à condition de pouvoir ensuite porter les modèles d'une plateforme de simulation à une autre.

Le logiciel de hAMSter, (racheté par Ansoft pour l'intégrer à Simplorer), a quant à lui une très grande simplicité d'utilisation, mais une implémentation plus incomplète de la norme et un moteur de simulation moins rapide. Il est cependant gratuit, ce qui en fait un concurrent redoutable et une base très intéressante pour l'enseignement. Le noyau de simulation de hAMSter a servi de base de développement pour celui du simulateur dernière génération d'Ansoft : Simplorer.

IV.2.2. Les nouveaux simulateurs disposant d'une GUI

Ces outils (Simplorer 6/7, SystemVision, …) représentent la dernière génération de simulateurs. Nés des besoins exposés ci-dessus, ils incluent la prise en charge et la cohabitation de ces différents langages afin de couvrir au mieux le spectre des niveaux et des types de modélisation existant. Le pionnier dans ce domaine, est Simplorer (Ansoft).

Cet environnement de simulation révolutionnaire, basé, pour sa partie VHDL-AMS, sur le noyau de simulation de hAMSter, permet d'allier une représentation graphique conviviale et intuitive à des modèles aussi complexes qu'on le souhaite (figure IV.2). De plus, dans sa version complète, le logiciel permet de créer des modèles C/C++ et de les interfacer, comme Matlab/Simulink ou Mathcad, avec VHDL-AMS. Ce logiciel inclut également la compatibilité avec d'autres outils de simulation du groupe Ansoft, comme "Maxwell" qui permet de modéliser des comportements avec une méthode de type éléments finis et de les intégrer dans un modèle système. [22] [28]

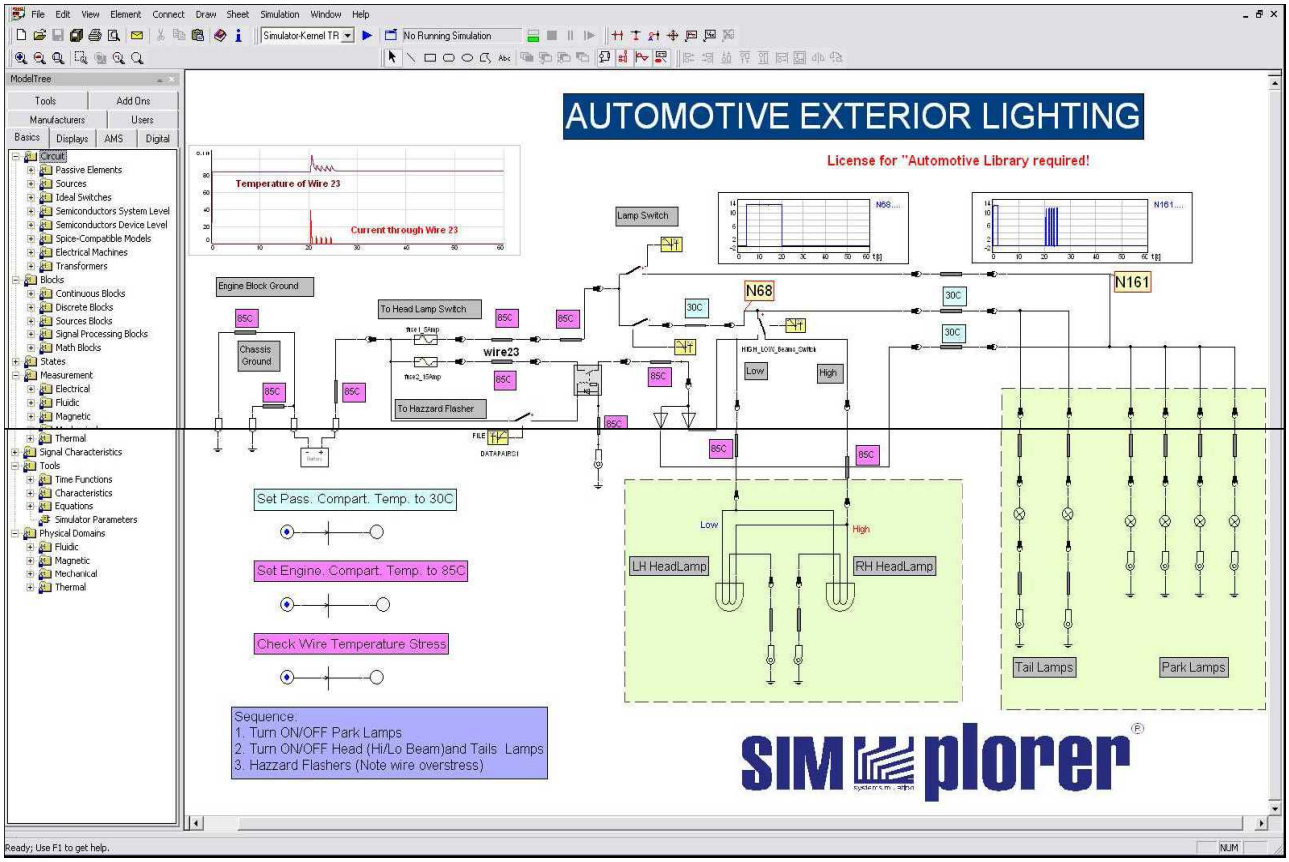


Fig IV.2:Vue schématique d'un circuit multi-abstractions,multi-domaines dans Simplore

IV.3. Simplorer

IV.3.1. Introduction

Ce simulateur appartenant à la dernière génération utilise un noyau de simulation VHDLAMS issu du simulateur hAMSter [9], disponible jusqu'alors gratuitement sur Internet. Ce logiciel supporte l'interfaçage entre les langages et logiciels tels que VHDL, VHDL-AMS, C, Simulink, Mathcad, et la suite de logiciels dédiés à la conception et à la simulation des systèmes électromagnétiques d'Ansoft. De plus, Simplorer utilise un langage propriétaire (SML) et qui lui permet d'utiliser une bibliothèque de primitive propres faisant intervenir des clones des modèles SPICE les plus utilisés et d'autoriser certains fonctionnements comme les réseaux de Petri. Son interface intuitive et riche en couleurs laisse une grande marge de manoeuvre tant au concepteur pour l'écriture des modèles et le dessin des symboles associés, qu'à l'utilisateur qui peut mettre facilement en avant les différents blocs de son schéma et y

incorporer directement les résultats de simulation qu'il souhaite afficher, rendant ainsi la feuille de schéma utilisable directement pour un rapport écrit.

Comme nous l'avons déjà écrit, c'est principalement à cause de sa sortie plus précoce que nous avons opté pour Simplorer, mais c'est également parce qu'il nous a apporté satisfaction que nous l'avons conservé.

IV.3.2 . L'interface graphique

L'interface graphique de Simplorer comporte plusieurs parties telles que les gestionnaires de symboles et de bibliothèques, ainsi que les interfaces de pré/post-traitements. Nous ne présenterons cependant ici (figureIV.3) que l'interface des feuilles de schémas permettant de connecter et de faire interagir les différents éléments d'un système :

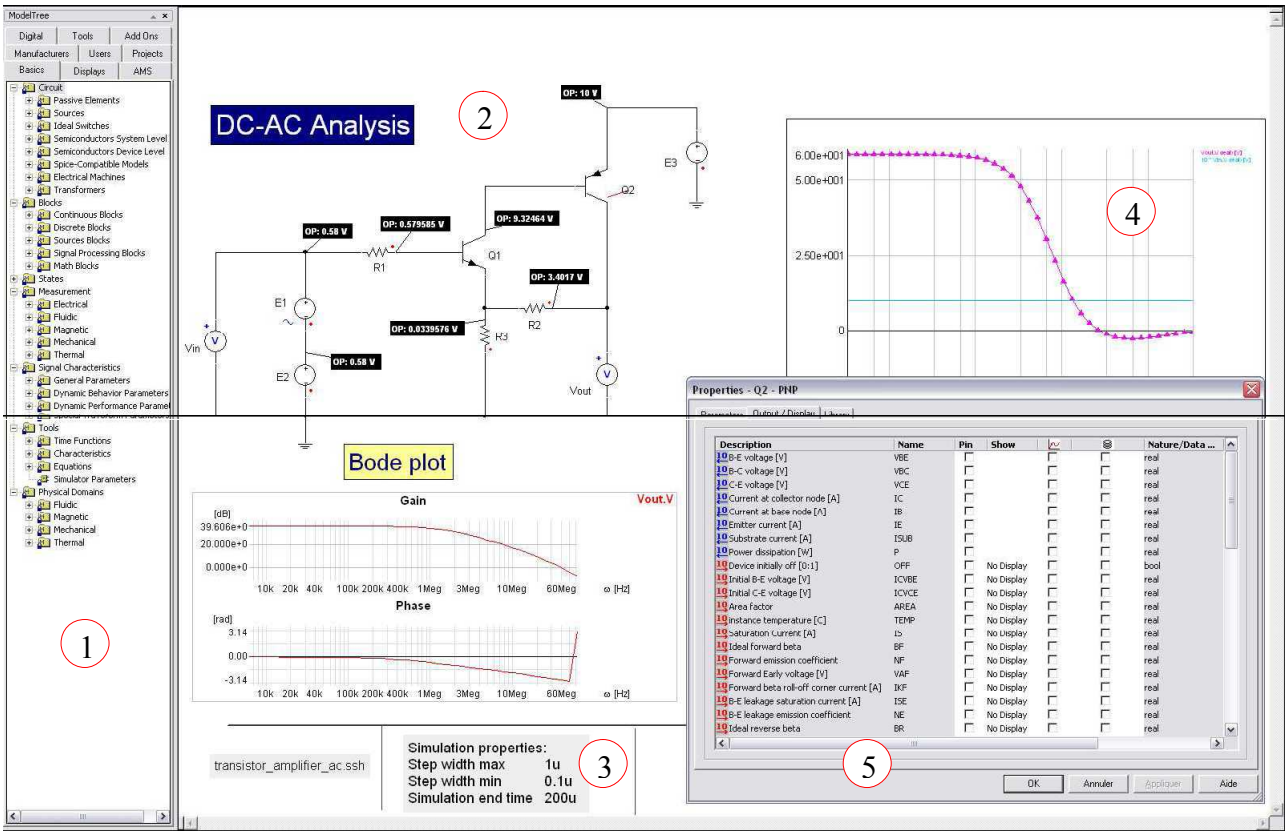


Figure IV.3 : Quelques aspects de l'interface graphique utilisateur.

Cette GUI met à portée de souris l'ensemble des composants disponibles dans les bibliothèques standard et importées, ainsi que dans celle propre au projet (work) (figure IV.3.1). L'ensemble de ces éléments est instanciable par simple "drag & drop" sur la feuille de schéma (figureIV.3.2). Ils peuvent ensuite être paramétrés selon différents aspects (couleurs, entrées/sorties, sélection des données pour le post-traitement, …) (figureIV.3.3).

Le logiciel offre également la possibilité d'afficher, au sein même de la feuille de schéma, différentes informations ou résultats de simulation comme les paramètres passés au simulateur (figureIV.3.4) ou des résultats de simulation bruts tels que les évolutions temporelles ou fréquentielles des signaux (figure IV.3.5).

Dans un environnement graphique comme celui de Simplorer, la représentation des composants devient un enjeu majeur de la compréhension par le plus grand nombre d'utilisateurs possible. Les gestionnaires de modèles et de symboles permettent en cela de leur associer des représentations graphiques, statiques, symboliques et explicites.

IV.3.3. La création de modèles en mode texte

Bien que la conception graphique apporte des solutions à un grand nombre de cas de figures ou permette d'avoir recours aux bibliothèques de composants adaptées, sans être confronté à leur code VHDL-AMS, elle ne résout pas tout. [28]

En effet, il faut bien créer les modèles utilisés dans les bibliothèques et donner aux spécialistes des langages le moyen de concevoir les objets qui ne peuvent pas être construits par l'assemblage d'éléments existants.

A cette fin, Simplorer dispose d'un éditeur de modèles en mode texte qui permet de définir les composants avec une certaine facilité. Il offre la possibilité de créer les différents aspects d'un composant au sein d'une même interface (figure IV.4).

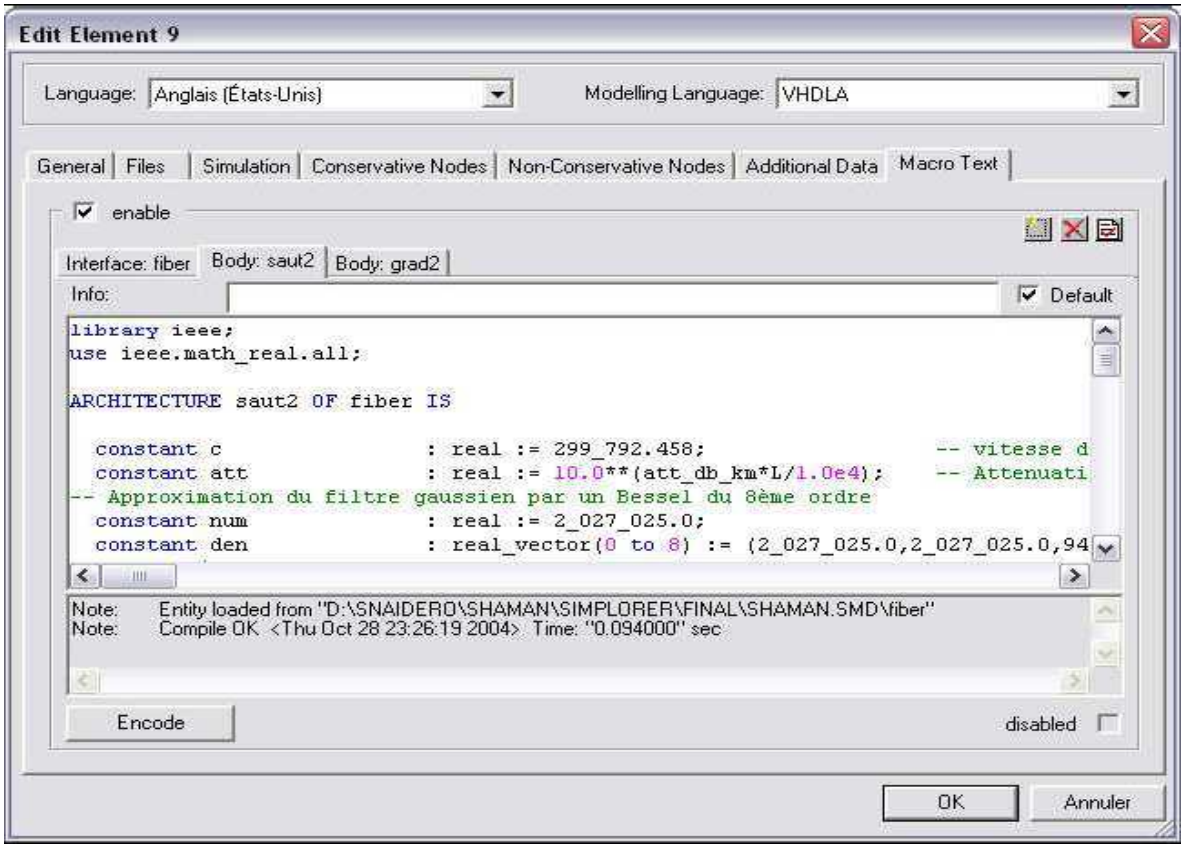


Figure IV.4 : Interface de conception textuelle des modèles.

On y trouve notamment la possibilité d'écrire des modèles en VHDL-AMS ou SML [5]. Les onglets permettent quant à eux de gérer les différents aspects d'un modèle (nom, symbole, fichiers impliqués, entrées/sortie, code). En ce qui concerne la fenêtre de code, elle comporte différents boutons pour créer/supprimer l'"ENTITY" et des "ARCHITECTURES" qui sont organisées de manière ingénieuse en onglets, et vérifier que le code écrit peut être compilé par Simplorer. La fenêtre comporte également un bouton "ENCODE" pour compiler le modèle au sein de la bibliothèque Simplorer. Ceci permet d'empêcher l'accès au code source du modèle.

IV.4. Application de la modélisation comportementale VHDL-AMS

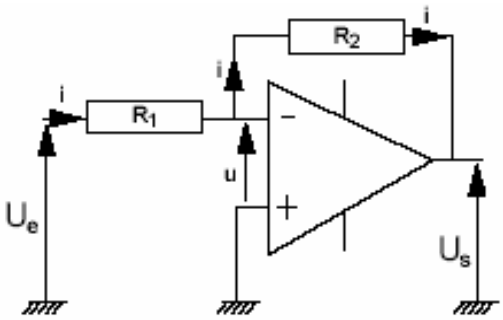
Dans la suite de ce chapitre, nous présentons la modélisation comportementale vhdl-ams d'un circuit réel, composant de dix huit transistors : un amplificateur inverseur. Sera ensuite présenté son implémentation dans le simulateur Simplorer.

Le circuit faisant l'objet du développement de modèle est considéré comme bloc fonctionnel. Il s'agit d'un amplificateur inverseur à base de 741 réalisé à base de transistors "monochip" et suivant le prototype de la compagnie Ferranti. Ce circuit est traité comme une boite noir, c à d seulement le comportement externe du circuit est d'intérêt primaire. Dans ce cas-ci, un ensemble de caractéristiques d'entrée, de transfert et de sortie est exigé. [4]

Une approche de Piece-Wice Linéaire (P.W.L) est alors appliquée pour approcher le comportement non linéaire de ces circuits à des régions linéaire chacune décrite par un circuit équivalent.Des équations linéaires sont jointes au modèle pour décrire chaque région d'opération, les éléments du modèle sont des impédances, et des sources de tension d'entrée et de sortie.

IV.4.1. Description du circuit amplificateur inverseur

L'amplificateur inverseur est le montage de base à amplificateur opérationnel, l'entrée non inverseuse est à la masse, le signal d'entrée est relié à l'entrée inverseuse par une résistance R1, comme la sortie est reliée à cette entrée par une résistance R2, comme le montre la figure . [4] [25]



FigureIV.5 : Schéma de l'amplificateur inverseur.

La caractéristique de transfert d'un amplificateur inverseur a pour expression :

$V_s = -(R_2/R_1)\, V_e$ pour $V_{sat}^- < V_s < V_{sat}^+$.

Dans ces conditions, le gain G_v de l'amplificateur est également déterminé par le rapport R_2/R_1 et s'écrit : $G_v=-(R_2/R_1)$.

L'amplificateur inverseur présente une résistance de sortie faible à cause de la contre réaction de tension appliquée à l'AOP. En revanche, sa résistance d'entrée est égale à R1.

La figure IV.6 représente le schéma de l'amplificateur opérationnel du commerce très couramment utilisé et connu sous la désignation "741".

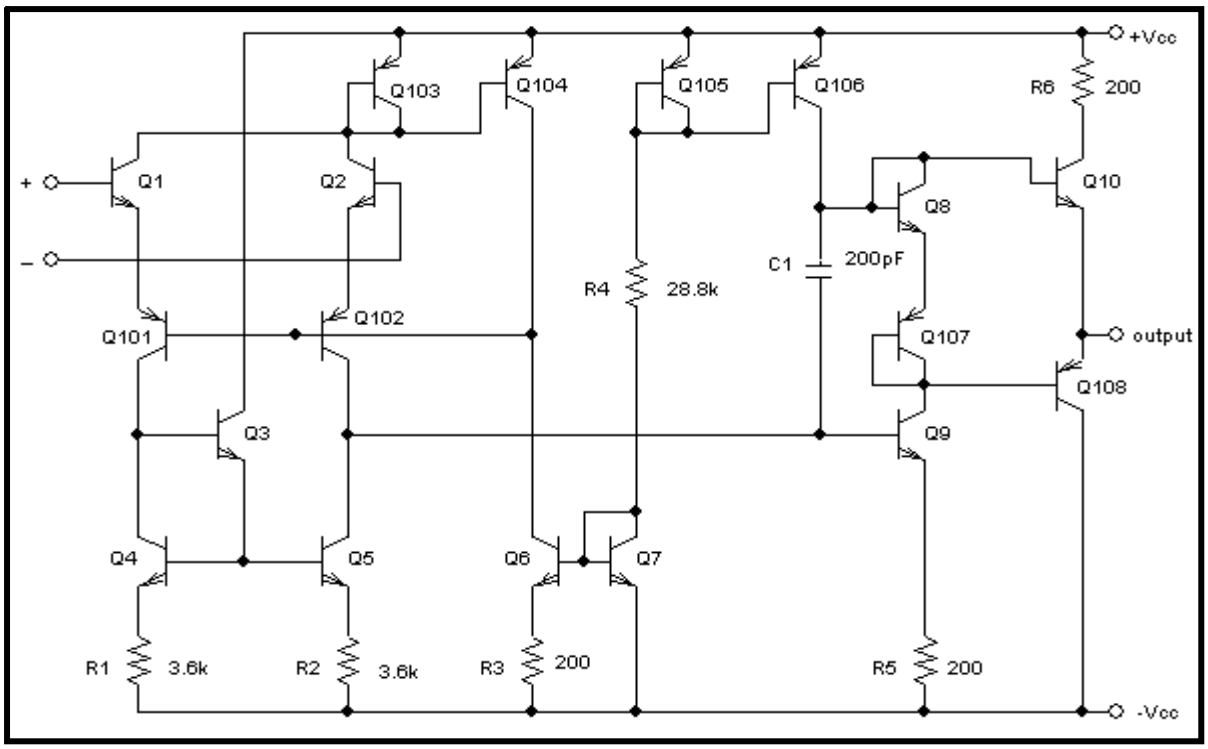


Figure IV.6 : Schéma interne de l'amplificateur opérationnel µA741. [12]

D'après ce schéma électrique, on peut distinguer et identifier trois étages associés en cascade :

IV. 4.1.1. L'étage d'entrée

Constitué par l'ensemble des transistors Q1, Q2, Q101, Q102, avec un fort gain par la suite de leurs associations, le gain en sortie de l'ensemble est important grâce aux charges actives constituées par les transistors Q4 et Q5. La polarisation de l'étage d'entrée est réalisée par le transistor monté en diode, Q103, et la source de courant Q104. Le miroir de courant fourni par cette source est maintenu constant (autour de 180µA), par un réseau polarisé construit avec les transistors Q6, et Q7.

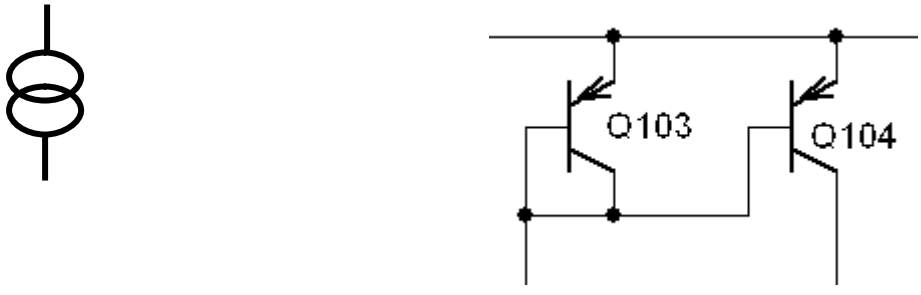


Figure IV.7 : Polarisation de l'étage d'entrée (Q103 monté en diode).

IV.4.1.2. L'étage intermédiaire (fondamental)

L'étage d'entrée est suivi par un circuit fondamentale (émetteur commun) à fort gain en tension, dont la sortie n'est plus différentielle, il est identifiée par la liaison du collecteur du Q102, et la base du Q9, son courant de polarisation est assuré par le générateur du courant des deux transistors Q105, et Q106, générant ainsi le courant miroir (650µA).

IV.4.1.3. L'étage de sortie

Utilise deux transistors montés en push-pull Q10 et Q108.

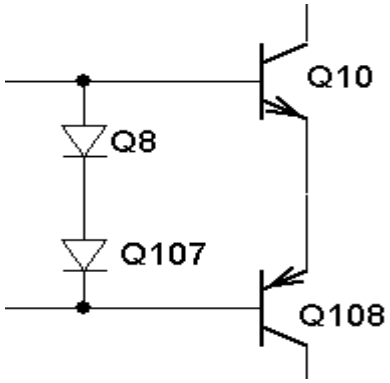


Figure IV.8 : Polarisation de push-pull.

Les deux transistors montés en diodes (Q107 et Q8) sont employés pour polariser les deux anciens transistors par environ 1.2 volts, c'est-à-dire elles permettent d'imposer la tension d'entrée de cet étage qui est en fait la tension de sortie de l'étage intermédiaire constante à une certaine valeur. En plus des éléments essentiels décrits ci-dessus, on trouve une capacité C qui régit le comportement de l'amplificateur opérationnel en fonction de la fréquence. Elle permet la suppression des composants continus quand il s'opère en régime alternatif. [15]

IV.4.2. Description du banc de mesure

Le 741 monté en inverseur, alimenté avec des tensions continus de ±5V, a donné de très bons résultats. Ce fut après le réglage de l'offset qui avait sans doute une grande importance. C'était aussi d'établir un minimum de quelques millivolts en applicant le 0 V à l'entrée(pratiquement V_{off}=8.87mV). On a choisi un gain de 4.7 en prenant R_1=1K et R_2=4.7K.

L'automatisation des mesures a été faite par l'exploitation de la carte IEEE488, le banc de modélisation représenté à la figure IV.9 consiste à :

- Installer la carte IEEE488.
- Installer le multimètre et le calibrateur.
- Ecrire des programmes en TURBO PASCAL, qui permettent la commande et l'acquisition des données, puis faire stocker ces derniers dans des fichiers pour un éventuel traitement.

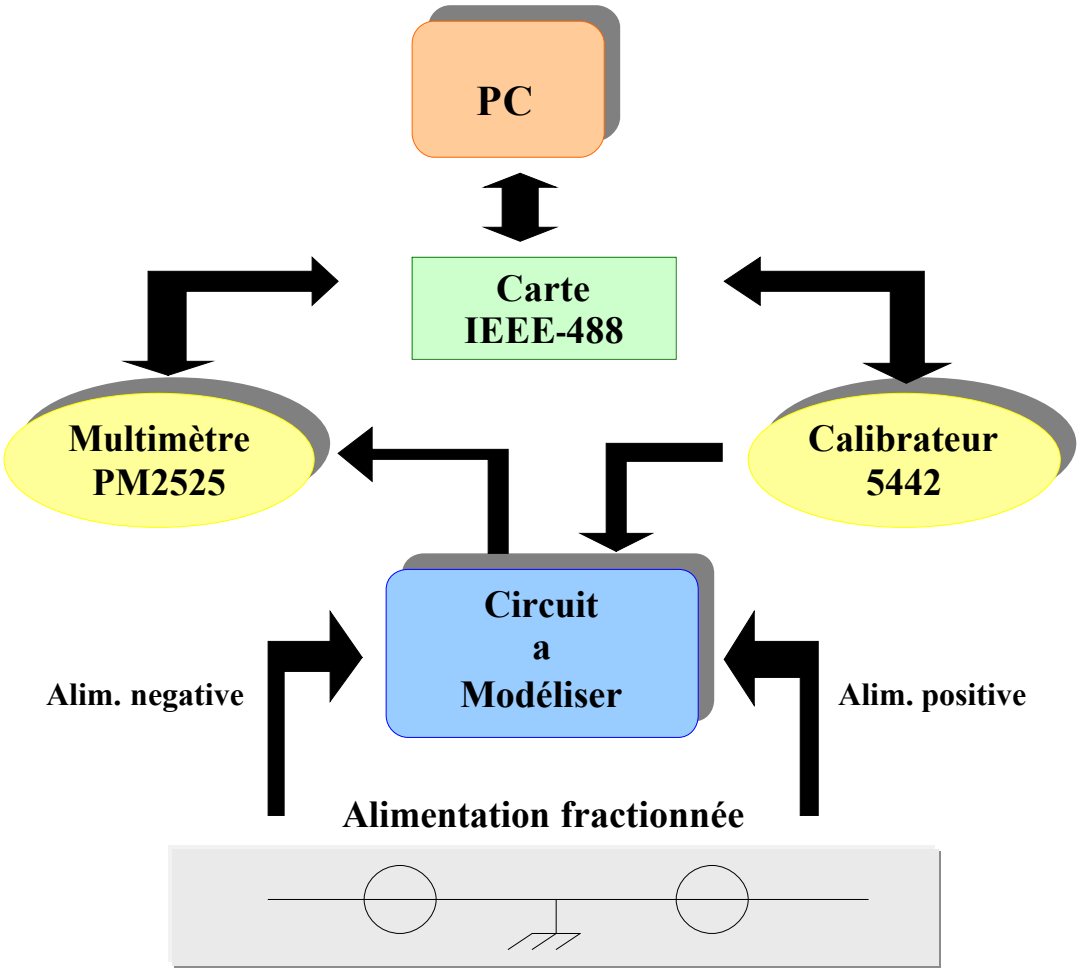


Figure IV.9 : Banc de modélisation. [4]

Profitant des acquis et résultats de la thèse [4], qui constitue pour nous une base concrète d'une méthodologie de conception de modèles à plus haut niveau grâce à l'utilisation de la technique célèbre : Piece-Wise Linear , notre travail porte sur cette technique.

IV.4.3. Piece-Wise Linear Technique

Notre technique proposée ici, consiste au développement de modèle de circuits analogiques en un circuit équivalent simple, uniforme et linéaire qui diffère totalement du circuit original mais reproduisant sa fonction non linéaire avec un degré de précision acceptable. D'ailleurs, ce modèle de circuit est établi à toutes ces régions de fonctionnement et un ensemble d'équations mathématiques linéaires est joint à lui décrivant son comportement. [4]

L'approche P.W.L développée dans notre travail est appliquée sur des caractéristiques de ports terminaux (de transfert, d'entrée et de sortie) obtenues à partir des expériences et des résultats de simulation. Dans cette approche, on peut visualiser la caractéristique terminale composée de plusieurs régions distinctes, dus aux paramètres non linéaires, qui changent de valeurs, et qui se résume en fin de compte à un ensemble de régions linéaires.

L'exécution de la P.W.L permet de délimiter les régions linéaires par des points de cassure (break points) ou points d'inflexion de la courbe des différentes Caractéristique d'un circuit et d'extraire les paramètres électriques composant un modèle électrique linéaire.

IV.4.3.1. procédure générale [4]

La procédure à suivre dans l'analyse par segmentation linéaire (P.W.L) est :

1. Déterminer tous les paliers significatifs (les points de cassures).
2. Faire toutes les approximations de simplification qui donne le degré d'exactitude exigé.
3. Employer ces approximations pour déterminer des paramètres terminaux et pour définir les régions entre les paliers par des équations mathématiques.

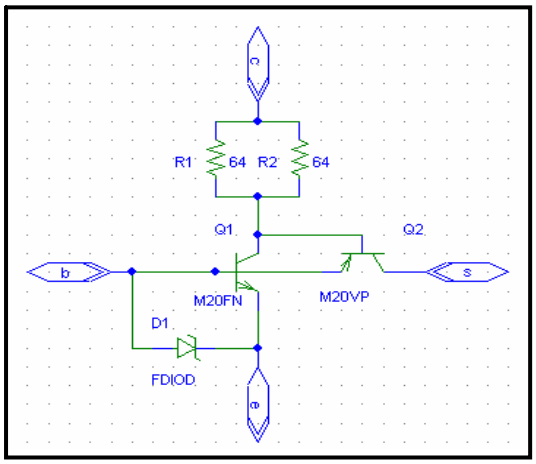
Cette technique est basée sur les caractéristiques de transfert, d'entrée et de sortie du circuit. Le format représentant ses caractéristiques terminales est discuté dans les sections suivantes.

IV.4.4. Simulation de l'amplificateur inverseur avec Pspice

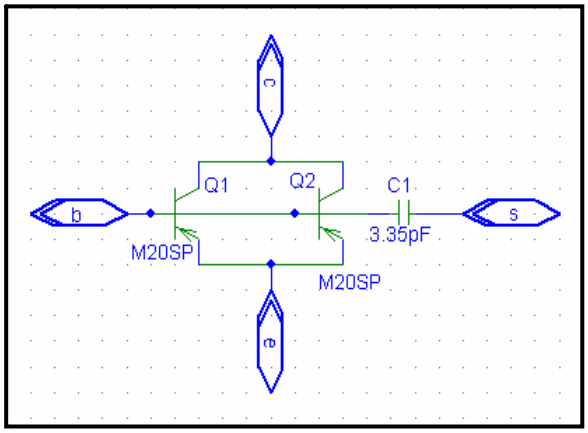
L'implémentation du circuit dans le simulateur PSPICE n'est pas si facile qu'on le croit mais plus tôt nécessite une attention bien particulière : le transistor composant cet amplificateur possède un modèle approprié et beaucoup plus complexe que celui offert par la bibliothèque du PSPICE. La réussite du processus de simulation exige d'abord l'insertion de ce modèle dans ce simulateur puis son utilisation pour la construction du circuit suivant des étapes qu'on décrira brièvement à travers les sections suivantes de ce chapitre.

IV.4.4.1. Structure des transistors utilisés

Le schéma électrique de l'amplificateur opérationnel disposé en figure IV.6 est propre à la maison originale FERRANTI, en outre, la réalisation fut possible grâce à la disposition des transistors (voir Fig.IV.10) intégrés dans un même circuit (***Monochip***) et offrant les mêmes caractéristiques électriques que celles du 741 intégré.



(a) Transistor NPN



(b) Transistor PNP

Figure IV.10 : Structure des transistors de type « Mono Chip »

IV.4.4.2. Description des étapes de simulation

Quand le schéma d'amplificateur opérationnel µA741 utilisé se compose de 18 transistors (10 transistors de type NPN et 08 transistors de type PNP), l'utilisation des transistors « Mono Chip » cités précédemment rend le schéma d'amplificateur inverseur très complexe (36 transistors, 10 diodes, 28 résistances, 09 condensateurs et 03 sources de tensions plus les masses). Afin de réduire la taille du schéma de notre circuit à simuler, Pspice nous permet d'associer chaque schéma de transistor à un bloc et de l'enregistrer comme un composant dans la librairie de Pspice.

Les transistors **M20FN, M20VP, M20SP** et la diode Zener **FDIOD** (M20FN est de type NPN, M20VP et M20SP sont de type PNP) n'existent pas dans la librairie de Pspice, donc il faut les créer et les enregistrer dans la librairie.

IV.4.4.2.1. la création de symbole par Pspice

Pour illustrer cette création de symbole, on a choisit le M20FN relatif au transistor de type NPN, cette façon reste valable pour les deux autres modèles (le M20SP et le M20VP) ainsi que celui de la diode zener.

La création du transistor M20FN a été faite en utilisant le « **Symbol Wizard** ». La figure IV.11 représente la page du rédacteur de symbole de Pspice « **Symbol Editor** », elle est affichée après la

sélection du « **Edit Library** » dans le menu « **File** » de Pspice Schematic, il faut donc sélectionner « **Symbol Wizard** » dans le menu « **Part** » pour le commencer.

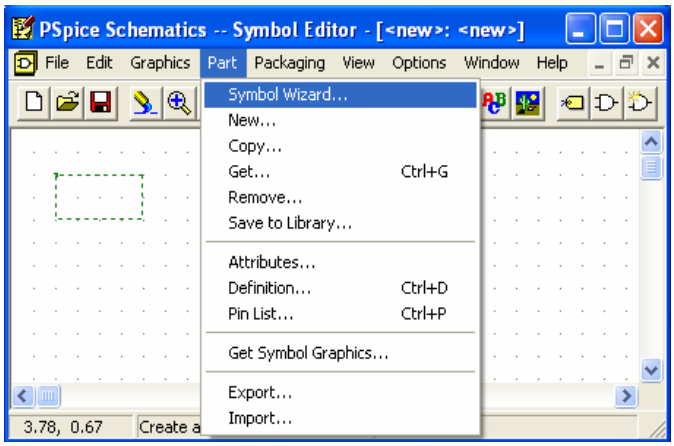
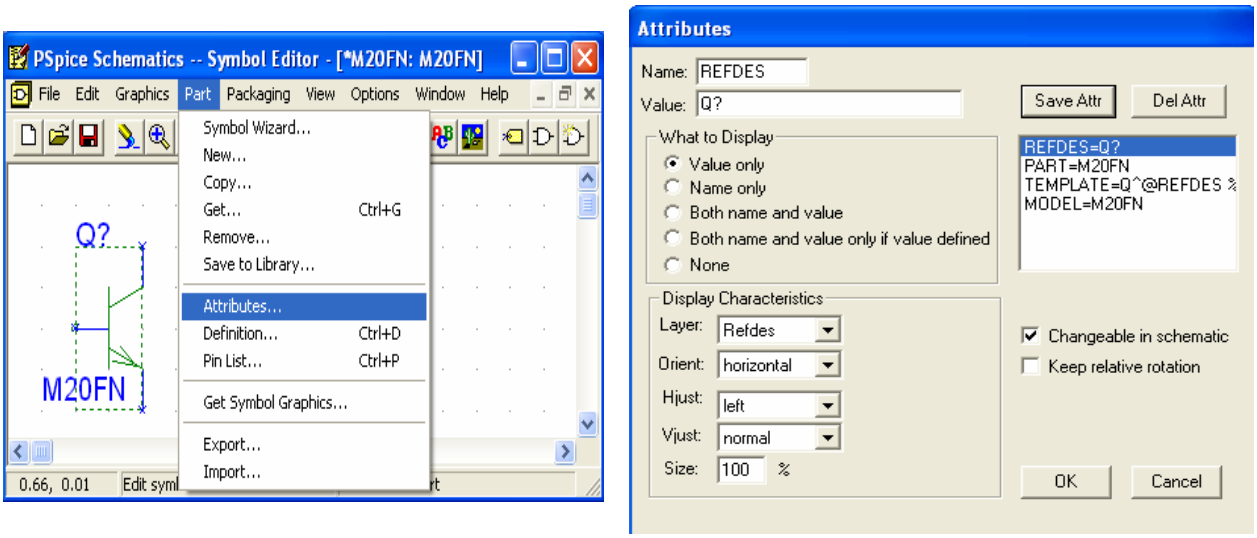


Figure IV.11 : Editeur de symbole dans Pspice.



(a) **(b)**

Figure IV.12 : Attribues du symbole.

L'insertion du modèle s'est faite en faisant introduire les valeurs des paramètres du modèle à l'aide de la fenêtre du bas de la boite de dialogue de l'éditeur de modèle donnée en figure IV.13. lors de l'édition du modèle dans la page de saisie de schéma, une copie de celui là est produite pour permettre des changements à introduire sur les instances des éléments spécifiques qui sont sauvegardés par défaut avec le même nom M20FN suivi de l »extension **.LIB**.

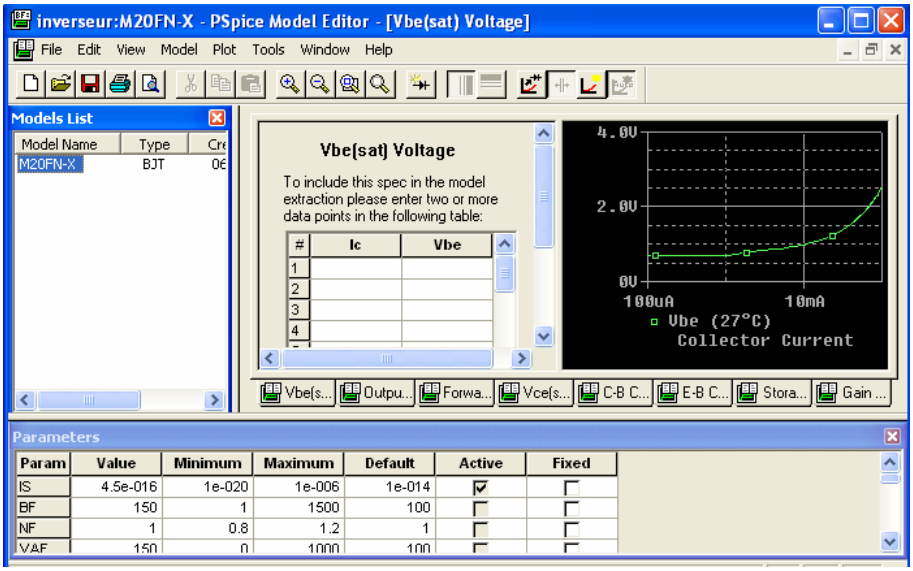


Figure IV.13 : Paramètres du modèle.

IV.4.4.2.2. Création des blocs: (hierarchical blocks)

Pour améliorer la taille du circuit à simuler, il faut créer des blocs comportant les schémas des transistors « Mono Chip » utilisés, après l'association des schémas aux blocs, ils devront être enregistrés dans la librairie de Pspice comme des symboles (composants).
Après l'édition de différentes caractéristique du bloc par l'éditeur de symbole, le bloc va se trouvé dans la librairie de Pspice sous la forme représentante dans la figure IV.14. il est près d'ètre utlisé comme tous les autres symboles d'origine du Pspice.

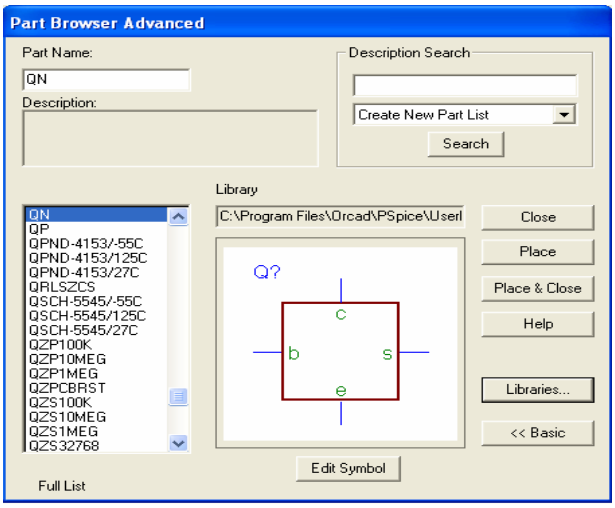


Figure IV.14 : Bloc sous forme de symbole.

Les autres composants (M20VP, M20SP et FDIOD) ont été créés de la même manière que le transistor M20FN.

IV.4.4.3. La simulation de l'amplificateur inverseur

Après le placement des composants, il faut les connectés en utilisant l'icône  (**draw wire**), puis attribuer les valeurs et les noms des composants, en cliquant deux fois sur le composant, la figure IV.15 montre le schéma à simuler par Pspice.

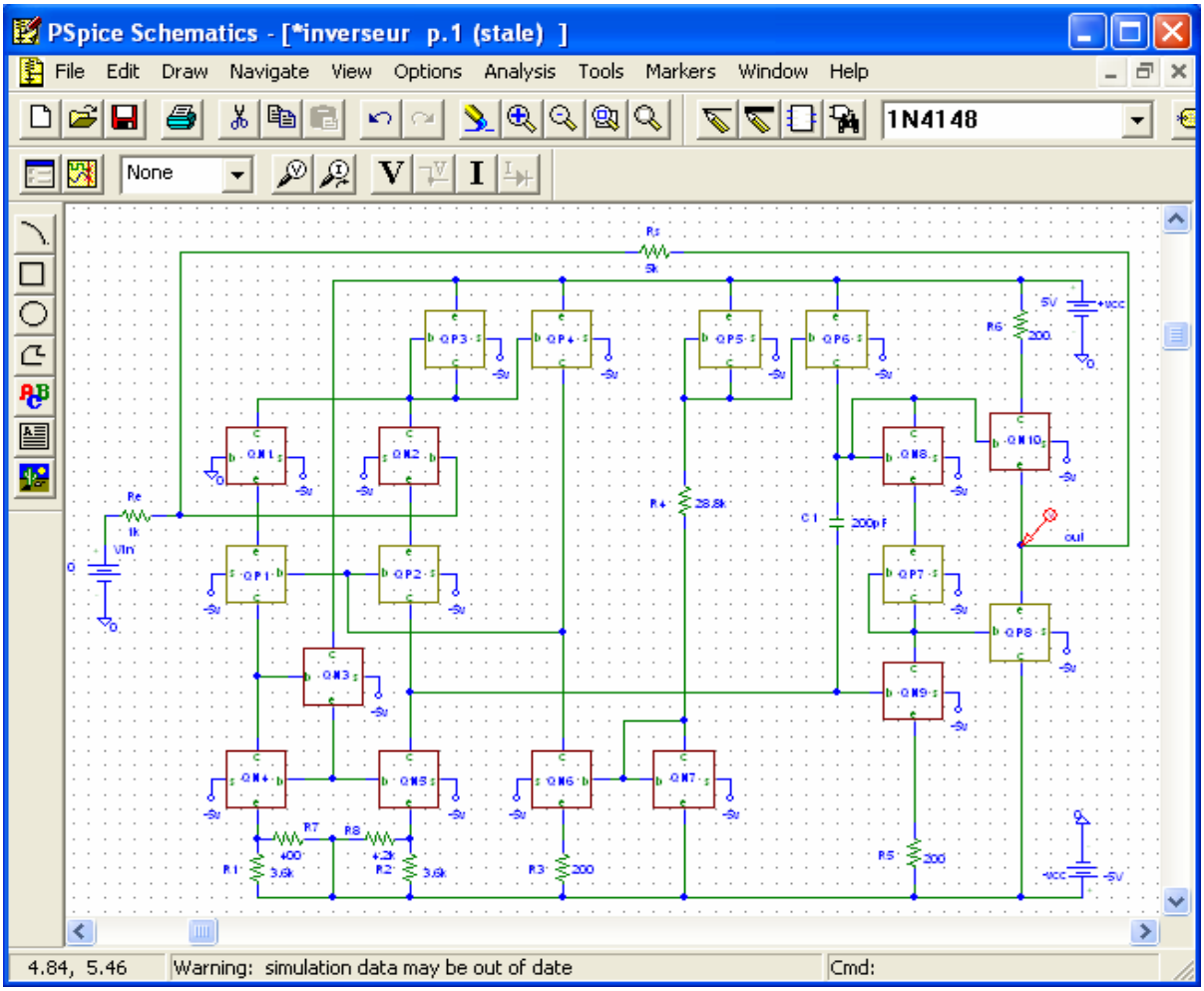


Figure IV.15 : Schéma d'amplificateur inverseur.

IV.4.4.3.1. Affichage des résultats de la simulation

Lorsque la simulation est terminée, la fenêtre « **Probe** » va ouvrir automatiquement, la figure IV.16 nous montre la caractéristique de transfert de l'amplificateur inverseur $V_s = f(V_e)$.

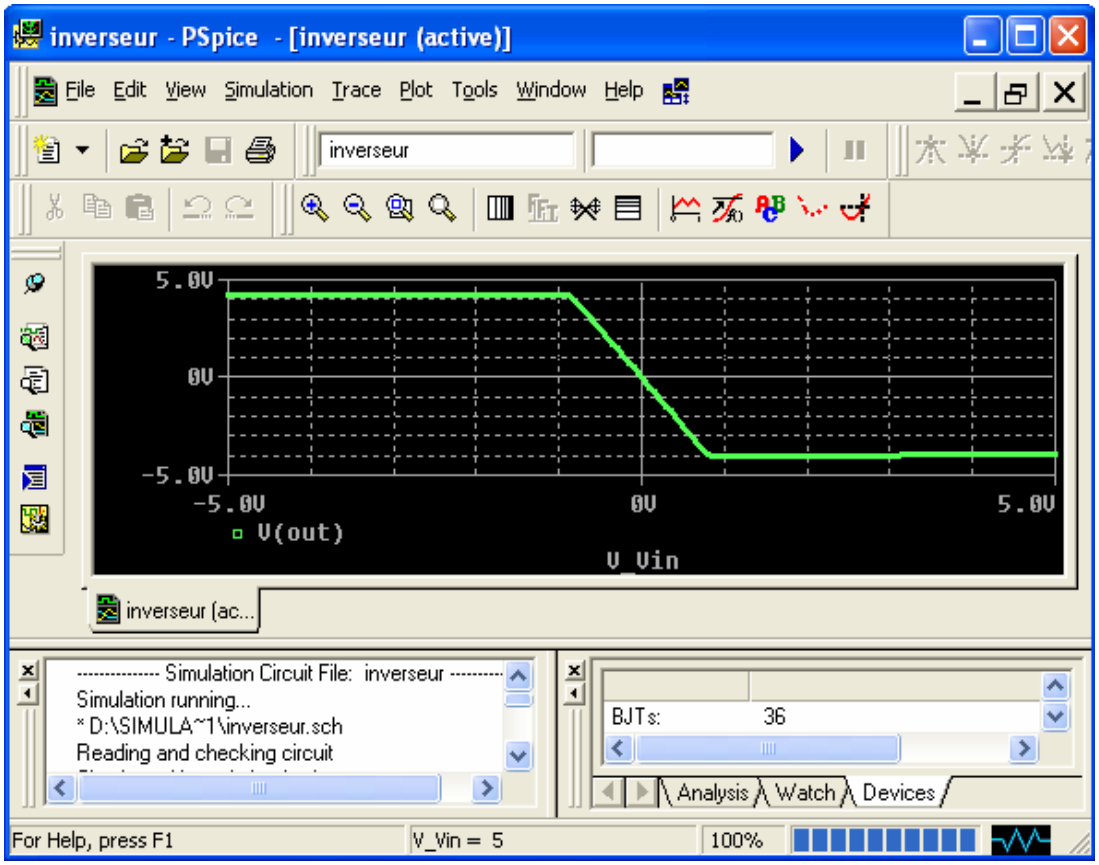


Figure IV.16 : Résultat de la simulation, caractéristique de transfert de l'amplificateur inverseur.

IV.4.5. Le modèle de P.W.L de l'amplificateur inverseur

L'amplificateur inverseur est alors vu comme une boite noire (voir fig IV.17), dont les éléments qui le composent sont inconnus et nécessitent d'être déterminés.

Le comportement du circuit est caractérisé par les trois régions d'opération: la région linéaire, et les régions de saturation positive et négative. Ceci est confirmé par les trois caractéristiques FigIV.19, FigIV.20, FigIV.21, qui serviront de données pour l'extraction des paramètres électriques à savoir : résistance d'entrée R_e, résistance de sortie R_s, source de tension contrôlée, …etc.

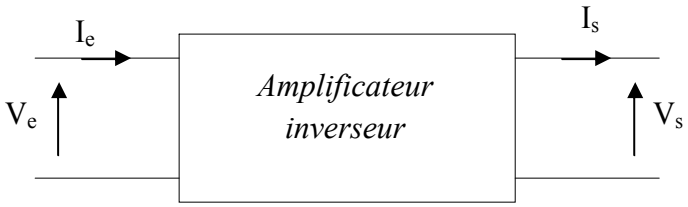


Figure IV.17 : représentation d'un amplificateur inverseur Comme une boite noire.

Un tel traitement rendra cette boite transparente et dont le circuit équivalent interne sera de la forme donnée en fig. IV.18.

Il s'agit d'un quadripôle à un port d'entrée et un port de sortie, les composants qui le constituent sont:

- **Comme paramètres d'entrée** :

  Une résistance d'entrée Re, et une tension résiduelle.

- **Comme paramètres de sortie** :

  Une résistance de sortie Rs, et une source de tension contrôlée ou non.

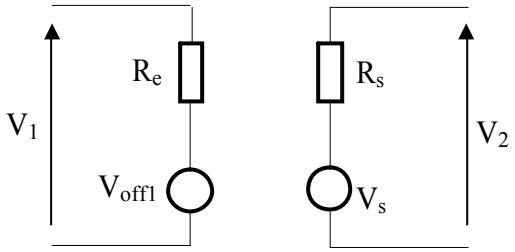


Figure IV.18: Circuit équivalent linéaire.

Pour un meilleur et simple développement du modèle de l'inverseur, on considère chaque région de la caractéristique d'une façon séparée.

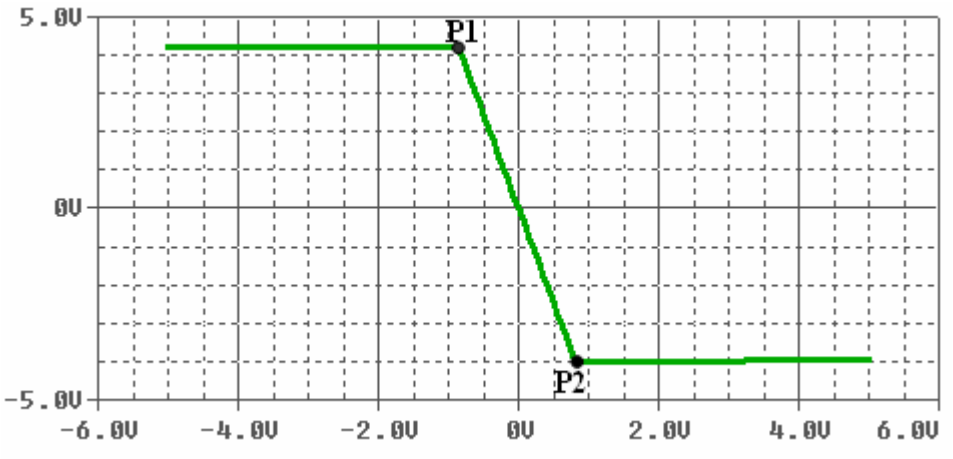


Figure IV.19 : Caractéristique de transfert Vs = f (Ve)

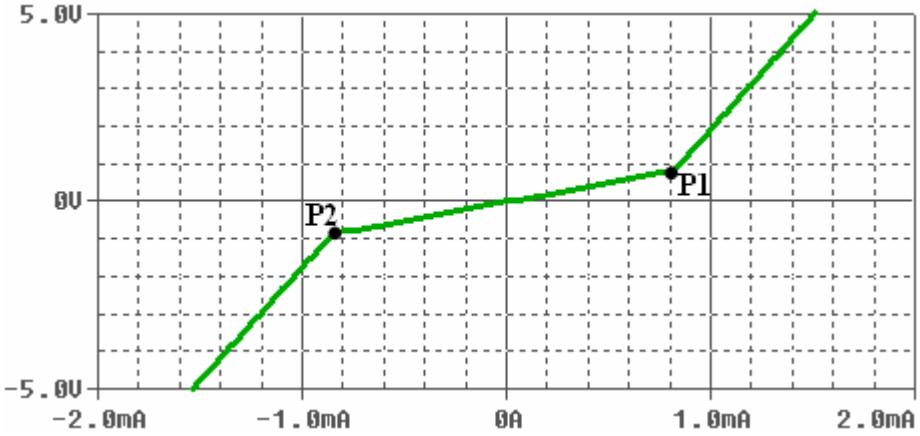
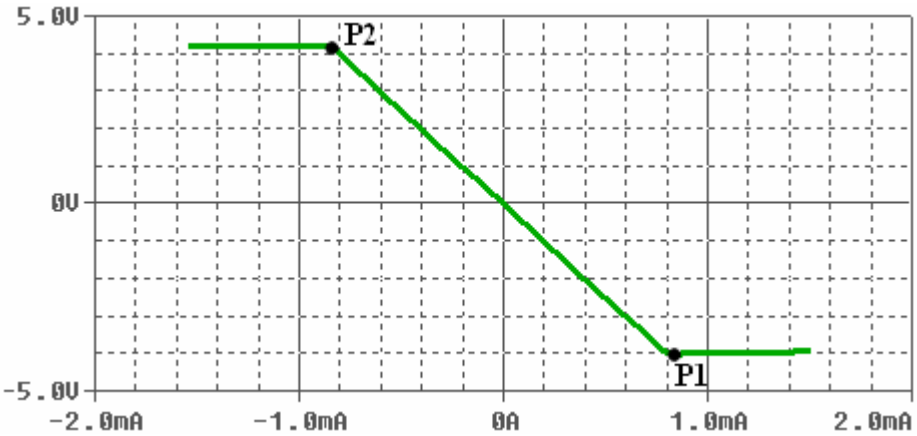


Figure IV.20 : Caractéristique d'entrée Ve = f (Ie)



Figur IVI.21: Caractéristique de sortie sans charge

IV.4.5.1. . Le modèle équivalent dans la région linéaire

A partir de différentes caractéristiques du circuit, les paliers sont ceux qui délimitent la région linéaire, ces points sont marqués comme **P1** et **P2** (voir les figures IV.19, IV.20, IV.21), et correspondent respectivement aux valeurs V_{e1}= -0.85V et V_{e2} = 0.85V de la tension d'entrée.

Entre ces deux valeurs, la caractéristique de transfert (Fig.IV.22) fournit une variation linéaire entre la tension d'entrée et la tension de sortie, ce qui conduit à un rapport constant comme étant le gain de l'amplificateur inverseur « **Gv** ». Il est évalué à partir de la pente de la région linéaire, et a été exactement déterminé par l'utilisation du programme d'ajustement de courbe (ORIGIN 6), et suivant un éventail de tension d'entrée allant de -0.85v à 0.85v (valeurs correspondantes aux points P1 et P2).

L'équation mathématique qui décrit cette région linéaire peut s'écrire comme telle :

$$V_s = AV_e + B \qquad \textbf{(IV.1)}$$

Le paramètre **A** exprime bien le rapport entre les tensions d'entrée et de sortie, et sa constance le long de la région illustre bien le régime linéaire de ce circuit.

Electroniquement, ce paramètre représente le gain en tension et peut se mettre sous la forme suivante :

$$A = G_V = \frac{\Delta V_s}{\Delta V_e} \qquad \textbf{(IV.2)}$$

D'autre part, le paramètre **B** est aussi constant. Mathématiquement il est déterminé comme étant V_s lorsque la tension d'entrée V_e est nulle.

Du point de vue électronique, ceci montre bien une tension e sortie non nulle quand celle de l'excitation à l'entrée est absente. Elle est due en réalité à la présence d'une tension résiduelle (tension d'offset) à l'entrée de l'amplificateur. Leurs valeurs sont alors comme suit :

$$V_s = AV_e + B \qquad \textbf{(IV.3)}$$

$$A = G_{Vlin} = -4.9513$$

$$B = V_{off\,2} = 0.0087V$$

a- Le port d'entrée équivalent

A travers la caractéristique d'entrée de la figure IV.23 , on peut déduire les paramètres d'entrée qui décrivent le comportement de l'amplificateur inverseur à son entrée. Ses éléments l'impédance d'entrée de l'inverseur, et la tension de décalage d'entrée (tension d'offset V_{off1}).

Pour des petits signaux à l'entrée (-0.85v à 0.85v), (voir fig IV.23), l'entrée de l'amplificateur inverseur se comporte comme une impédance R_e.

Selon la caractéristique d'entrée $V_e = f(I_e)$, le courant I_e et la tension V_e sont liées linéairement entre eux dans l'intervalle délimité par les deus points cités auparavant à savoir P1 et P2. Ils peuvent s'exprimer alors comme :

$$V_e = A' I_e + B' \qquad \textbf{(IV.4)}$$

L'évaluation de A' est faite à partir de la pente à la courbe tel que :

$$A' = \frac{\Delta V_e}{\Delta I_e} \qquad \textbf{(IV.5)}$$

C'est une résistance, sa valeur est trouvé égale à celle de R_1.

B' est obtenu en absence de courant (en court-circuitant la source de tension à l'entrée), c'est une tension d'offset présente à l'entrée de l'amplificateur, elle est très faible (quelques mV).

Donc,l'entrée de l'amplificateur inverseur est vue comme une résistance R_1, en série avec une source de tension constante $V_{off\,1}$ dont leur relation avec la tension d'entrée et leurs valeurs sont :

$$V_e = A' I_e + B' \qquad \textbf{(IV.6)}$$

$$A' = R_e = 1k\Omega$$

$$B' = V_{off1} = 0.991mV$$

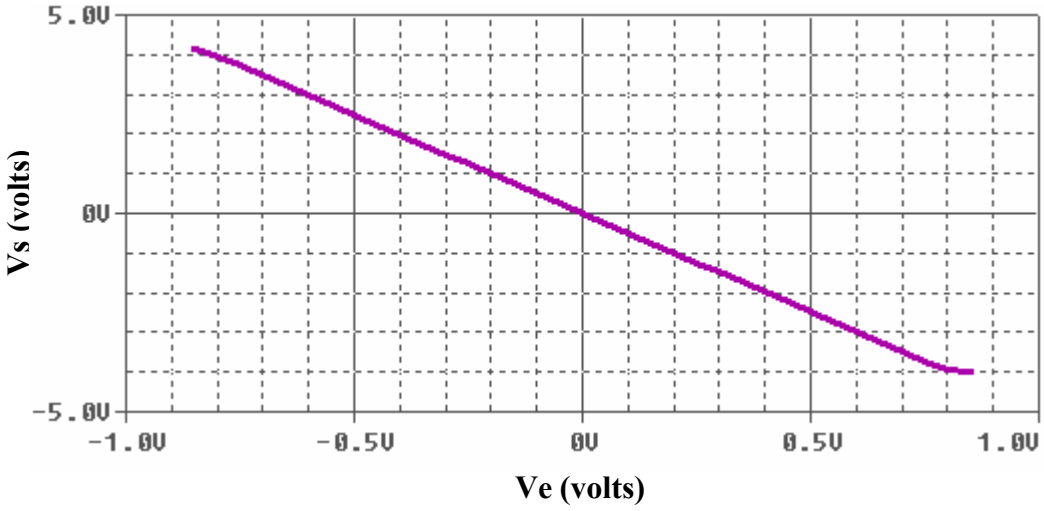


Figure IV.22: caractéristique de transfert (région linéaire)

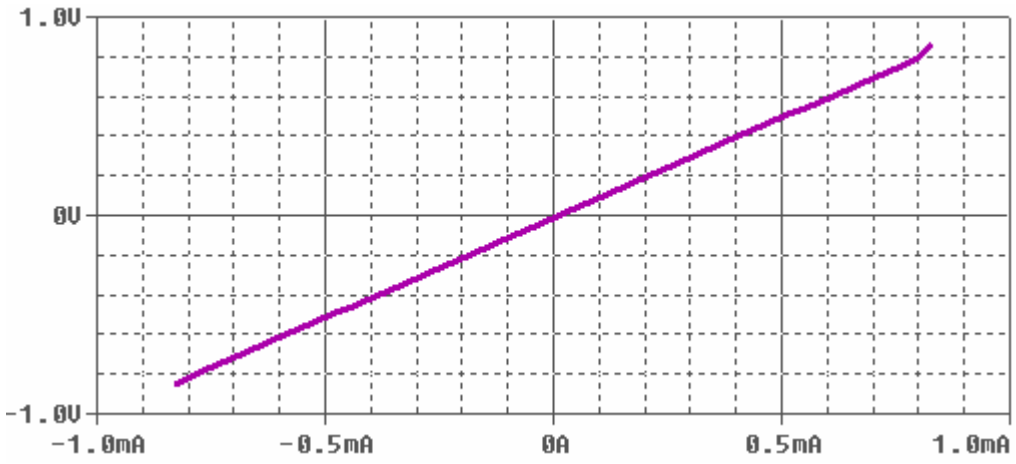


Figure IV.23 : caractéristique d'entrée (région linéaire)

b- Le port de sortie équivalent

L'étage de sortie de l'amplificateur inverseur est équivalent à une source de tension, en fait l'équivalence de Thévenin introduit une résistance série non nulle, c'est la résistance de sortie de l'amplificateur inverseur R_s.(voir fig.IV.24).

Pour déterminer la valeur de R_s, il faut tenir compte de la charge, et par le théorème de Thévenin appliqué au circuit, R_s est exprimée comme suit :

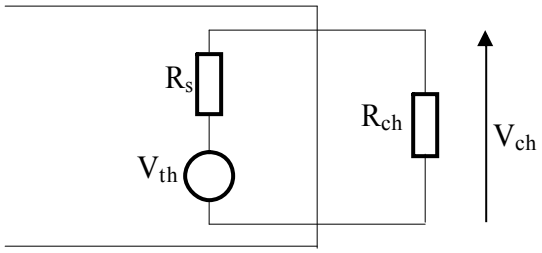


Figure IV.24 : Circuit de Thevenin du port de sortie de l'amplificateur inverseur

$$R_s = \frac{V_{th} - V_{ch}}{I_{ch}} \qquad \textbf{(IV.7)}$$

Avec : $V_{th} = G_{vlin}.V_e + V_{off\,2}$ **(IV.8)**

Les valeurs sont de l'ordre de certains Ohms. Avec ce modèle, la tension de sortie n'est pas indépendante du courant de sortie I_s, elle décroît linéairement avec une pente R_s.

Un modèle complet de l'amplificateur inverseur fonctionnant dans sa région linéaire est donné dans le tableau IV.1, ceci inclut les équations mathématiques et les points finaux qui localisent cette région d'opération du circuit.

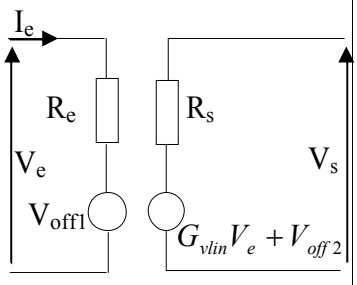
Région de fonctionnement	Equations	Valeurs des éléments	Modèle linéaire
-0.85 V < V_e < 0.85 V	* $V_s = G_{vlin}.V_e + V_{off\,2}$ * $V_e = R_e.I_e + V_{off\,1}$ * $R_s = (V_{th} - V_{ch})/I_{ch}$	G_{Vlin} = - 4.9513 R_e = 1 k V_{off1} = 0.991 mV V_{off2} = 8.7 mV	

Tableau IV.1 : Modèle de l'amplificateur inverseur dans la région linéaire.

IV.4.5.2. le modèle équivalent des régions de saturation

Le but de cette section est de développer des modèles pour l'amplificateur inverseur fonctionnant dans ses régions de saturation, ces modèles représentent les changements non linéaires des paramètres du circuit.

A partir d'un zooming sur les régions (en dehors des points P1 et P2) des différentes caractéristiques, il s'avèreque les tensions d'entrée et de sortie varient entre eux et avec les courants d'entrée (I_e) et de sortie (I_s) d'une façon linéaire jusqu'aux valeurs limites atteignant celles de la tension d'alimentation (+5V).

Une illustration graphique (les figures IV.25, IV.26,IV.27,IV.28) le confirme parfaitement.

Cette observation nous a conduit à établir un modèle linéaire pour chacune de ces régions (saturation positive, et saturation négative) tout en préservant les limites desquelles le modèle est validé : les points de cassures P1 et P2 et la tension de service qu'il ne faut pas dépasser.

La caractéristique de transfert mesurée (Fig.IV.26) prouve que les limites de tension de sortie pour les grandes valeurs de la tension d'entrée $|Ve|$>0.85V sont moins que les tensions d'alimentation (Vs<Vcc), cependant, ces niveaux limites ne sont pas tout à fait constants, la caractéristique indique un changement linéaire à faible pente de la tension d'entrée avec celle de la sortie.

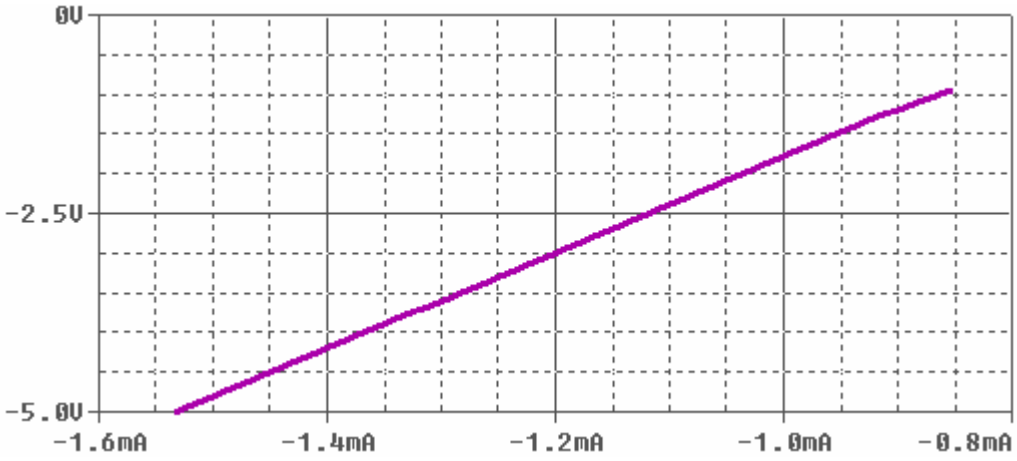


Figure IV.25 : caractéristique d'entrée (région de saturation positive)

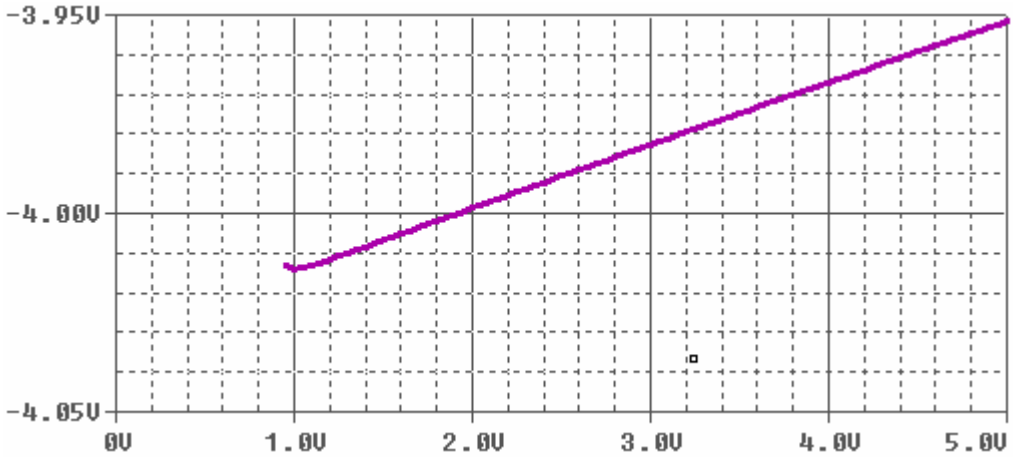


Figure IV.26 : caractéristique de transfert (région de saturation négative

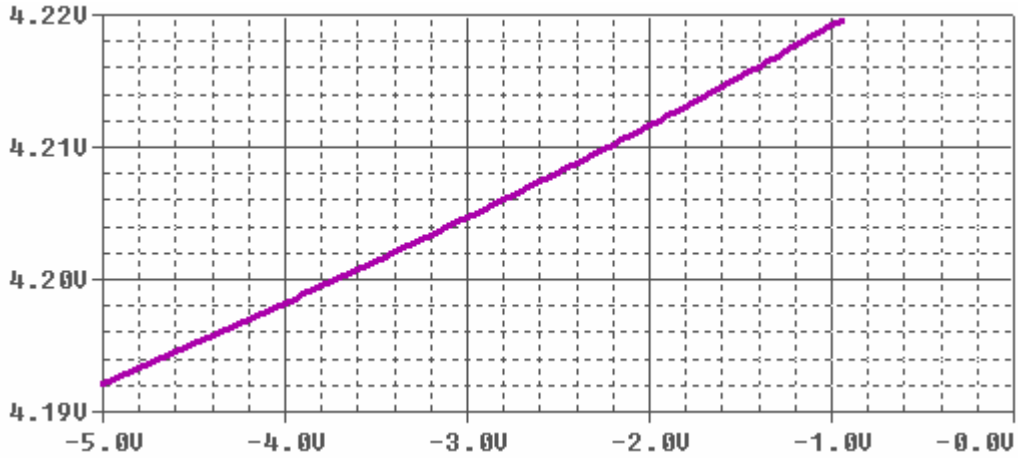


Figure IV.27 : caractéristique de transfert (région de saturation positive)

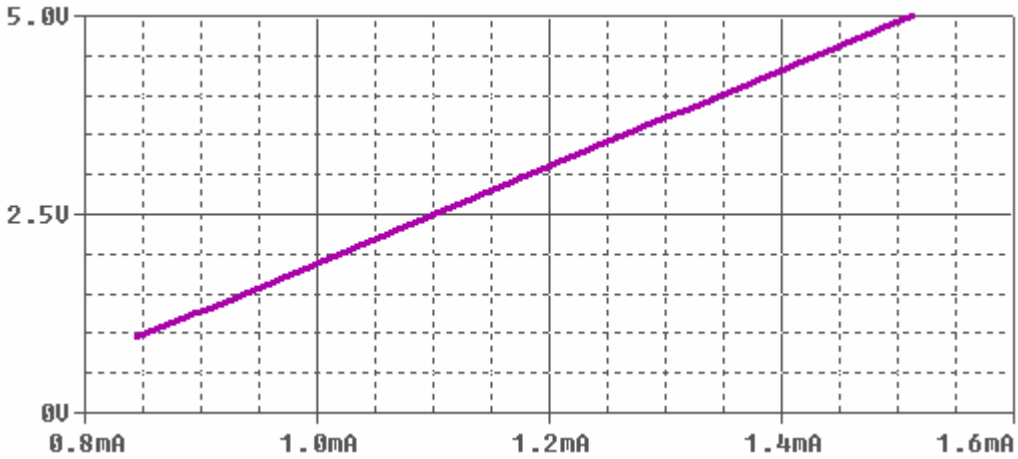


Figure IV.28 : caractéristique d'entrée (région de saturation négative)

a- Paramètres électriques en régions de saturation

Le port de sortie du circuit est vu en absence de charge comme une source de tension à valeur constante. Elle s'exprime comme suit :

$$V_s = AV_e + B \qquad \textbf{(IV.9)}$$

Où a est le gain à faible valeur, et b est une tension constante.

En terme électronique, a (gain très faible) exprime la variation légère mais d'une façon linéaire de la tension de sortie et celle d'entrée. Quant au b , c'est la tension de saturation positive proche de celle de la tension d'alimentation, leurs valeurs et leurs relations avec V_e et V_s sont :

En saturation positive, et sur un intervalle de tension entre -4V et -0.85V

$$V_s = aV_e + b \qquad \textbf{(IV.10)}$$

$$a = G_{Vsp} = 0.00524$$

$$b = V_{2sp} = 4.2375V$$

En saturation négative, les deux valeurs s'obtiennent en considération tant la région dont la gamme de tension est de 0.85V à 4V.

$$V_s = a'V_e + b' \qquad \textbf{(IV.11)}$$

$$a' = G_{Vsn} = 0.01584$$

$$b' = V_{2sn} = -4.123V$$

Nous remarquons une différence de valeurs nette entre les paramètres dans les deux régions et que nous pensons qu'elle est due à une asymétrie dans les caractéristiques électriques des transistors employés.

b- Le port d'entrée équivalent en régions de saturation positive et négative

Analogiquement à la région linéaire,les deux régions de saturation présentent des courbes linéaires,(figures IV.25,IV.28). l'expression de la tension d'entrée V_e s'écrit comme :

$$V_{es} = a_s^{'} I_e + b_s^{'}$$

(IV.12)

Où $a_s^{'}$ et $b_s^{'}$ sont des coefficients qui dépendent de la région de saturation (positive ou négative) et qui correspondent électriquement et respectivement à une résistance R_{es} et une source de tension indépendante V_{1s}.

Leurs valeurs sont déterminées comme la pente à la courbe pour R_{es} et l'extrapolation de la courbe pour V_{es} (lorsque le courant I_e est nul).

- Région de saturation positive :

$$V_e = a_{sp}^{'} I_e + b_{sp}^{'}$$

(IV.13)

$$a_{sp}^{'} = R_{esp} = 5.92k\Omega$$

$$b_{sp}^{'} = V_{1sp} = 4.27V$$

- Région de saturation négative :

$$V_e = a_{sn}^{'} I_e + b_{sn}^{'}$$

(IV.14)

$$a_{sn}^{'} = R_{esn} = 5.97k\Omega$$

$$b_{sn}^{'} = V_{1sn} = -4.16V$$

c- Le port de sortie équivalent en régions de saturation

En régime de saturation, l'objectif est double, il est indispensable de déterminer la valeur de la résistance de sortie R_s de l'amplificateur inverseur sans charge à la sortie et l'influence de la charge sur le paramètre.

L'évaluation s'est faite expérimentalement en insérant l'ampèremètre entre la sortie du circuit inverseur et la charge. en appliquant le théorème de thévenin pour les deux cas de régions de saturation dont les gammes de tensions d'entrée appropriées sont citées au préalable.

La valeur de cette résistance est déterminée par l'expression suivante :

$$R_s = \frac{V_{th} - V_{ch}}{I_{ch}} \quad \textbf{(IV.15)}$$

La région de saturation (positive ou négative) est caractérisée par une courbe linéaire, ce qui nous a résulté une résistance de sortie R_s constante sur toute la gamme de tension d'entrée limitant la région.

Les modèles complets de l'inverseur fonctionnant dans ses régions de saturations positive et négative sont donnés dans le tableau IV.2.

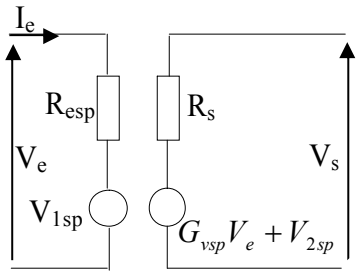
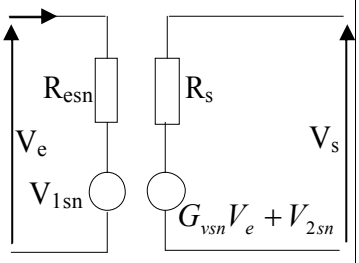
Région de fonctionnement	Equations	Valeurs des éléments	Modèle linéaire
Saturation positive **$-5V < V_e < -0.85V$**	$*V_s = G_{vsp}.V_e + V_{2sp}$ $*V_e = R_{esp}.I_e + V_{1sp}$ $*R_s = (V_{th} - V_{ch})/I_{ch}$	$G_{vsp} = 0.00676$ $R_{esp} = 5.97\ k\Omega$ $V_{1sp} = 4.18V$ $V_{2sp} = 4.226V$	
Saturation négative **$0.85V < V_e < 5V$**	$*V_s = G_{vsn}.V_e + V_{2sn}$ $*V_e = R_{esn}.I_e + V_{1sn}$ $*R_s = (V_{th} - V_{ch})/I_{ch}$	$G_{vsn} = 0.01523$ $R_{esn} = 6.03\ k\ \Omega$ $V_{1sn} = -4.027\ V$ $V_{2sn} = -3.99\ V$	

Tableau IV.2 : Modèle de l'amplificateur inverseur dans les régions de saturation

IV.4.6. le modèle VHDL-AMS :

Il est maintenant possible à partir du modèle élaboré de développer le modèle VHDL-AMS de l'amplificateur inverseur, ce modèle est constitué d'une entité pour la déclaration des paramètres et des bornes de connexion, et d'une architecture pour la description des équations.

Le modèle présente une technique de modélisation permettant de gérer les paramètres du modèle (les valeurs par défaut en particulier).

Le code VHDL-AMS est le suivant:

```
library ieee_proposed;
use ieee_proposed.electrical_systems.all;
entity Ampli_nverseur is
generic ( R1:real:= 1000.0;
          R2:real := 4.7e3;
          Rin : real:= 1.0e3;
          Rout : real:=100.0;
          voff1 :real:=99.0e-5;
          voff2:real:=87.0e-4;
          Rinn:real:=5.97e3;
          Rinp :real:=5.92e3;
          voff1n :real:=-4.16;
          voff1p :real:=4.27;
          gainn :real :=5.24e-3;
          gainp :real :=15.8e-3;
          voff2n :real:=-4.123;
          voff2p:real:=4.237 );

port (terminal tin, tout, tref1,tref2: electrical);

------------------------------------------------------------------------
-- vérification des paramètres
------------------------------------------------------------------------
begin
assert Rin > 0.0
report "Erreur (E): Rin doit être > 0" severity error;
end entity Ampli_nverseur;

architecture model of Ampli_nverseur is
------------------------------------------------------------------------
-- calcul des paramètres internes
------------------------------------------------------------------------
constant gain :real:= -R2/R1;
------------------------------------------------------------------------
-- quantités
------------------------------------------------------------------------
quantity uin across iin through tin to tref1;
quantity uout across iout through tout to tref2;
begin
```

```
-------------------------------------------------------------------------
-- affichage des paramètres internes
-------------------------------------------------------------------------
process begin
report " parametres sont :" & LF&
       "gain =" & real'image(gain)&"a" &LF &
       "voff2 =" & real'image(voff2) & "V";
wait;
end process;
-------------------------------------------------------------------------
-- region de saturation positive
-------------------------------------------------------------------------
if not uin'above(-0.85)  use
uin==Rinp*iin+voff1p;
uout==Rout*iout+gainp*uin+voff2p;


-------------------------------------------------------------------------
-- region lineaire
-------------------------------------------------------------------------
elsif  uin'above( 0.85 ) use
uin == Rinn*iin+voff1n;
uout==Rout*iout+gainn*iin+voff2n;


-------------------------------------------------------------------------
-- region de saturation negative
-------------------------------------------------------------------------
else
uin ==Rin*iin+voff1;
uout == Rout*iout+gain*uin+voff2;
end use;
break on uin'above(0.85),uin'above(-0.85);
end architecture model;
```

La partie exécutable de l'entité contient en plus des instructions d'assertion permettant la vérification de valeurs de paramètres génériques pour éviter des non convergences en simulation ou des résultats non réalistes.

Le corps d'architecture déclare tout d'abord la constante Gain qui représente un paramètre interne du modèle dont les valeurs sont dérivées de celles de paramètres génériques. Les paramètres génériques représentent en général des spécifications que le modèle doit garantir, alors que les paramètres internes représentent en général les conséquences des spécifications. Il est toutefois possible de choisir une répartition différente entre paramètres génériques et paramètres internes.
Viennent ensuite les déclarations du terminal interne, des quantités de branches représentant les tensions d'entrée et les tensions de sortie de chaque région du modèle.

Le rôle de l'instruction **process** utilisé dans le modèle est d'afficher les valeurs des paramètres internes. Le processus ne s'exécutera qu'une seule fois grâce à l'instruction **wait**. On obtient l'affichage suivant avec les valeurs par défaut des paramètres génériques:

```
NOTE_SL     [Report]
        Info 100103  - <Report>  note :  PROCESS/PROCEDURAL Process0:  parametres
sont :
gain =-4.700000e+000
voff2 =8.700000e-003V
```

Les Figures IV.29 et IV.30 donnent le résultat de la simulation du modèle avec les paramètres mentionnés aux tableaux IV.1 et IV.2.

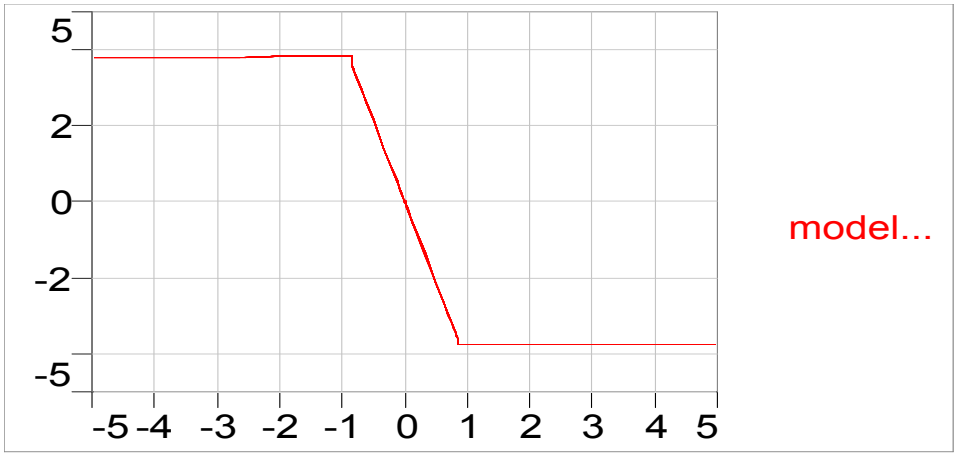


Figure IV.29. Simulation Simplorer du modèle (caractéristique de transfert).

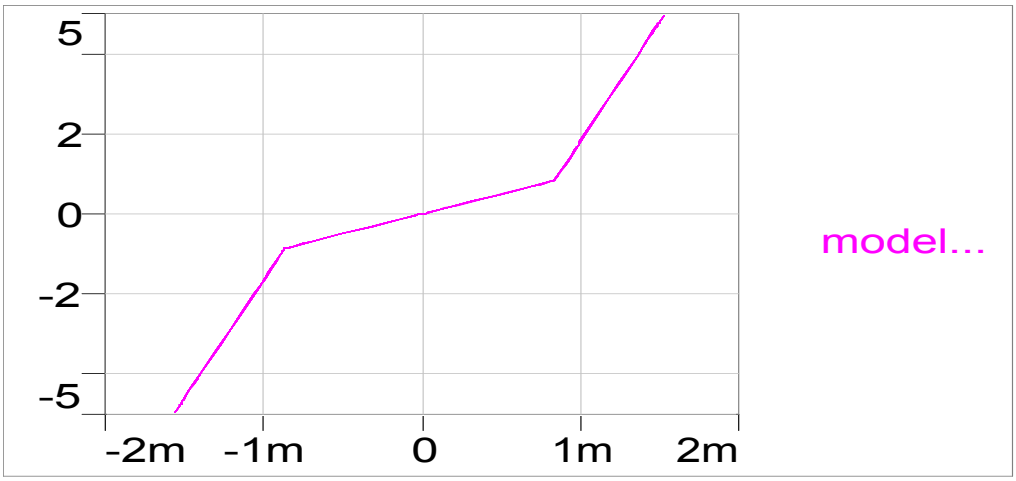


Figure IV.30 : Simulation Simplorer du modèle (caractéristique d'entrée).

On constate que le gain de $-R2/R1 = -4.7$ est bien obtenu.

Dans ce chapitre nous avons montré la modélisation et la spécification schématique en utilisant les potentialités du langage VHDL-AMS.

Grâce à Simplorer, l'outil de modélisation et de simulation VHDL , nous avons élaboré et validé le modèle de l'amplificateur inverseur . Nous avons par la suite rangé ce modèle dans la bibliothèque du logiciel Simplore pour la réutilisation.(Fig IV.31).

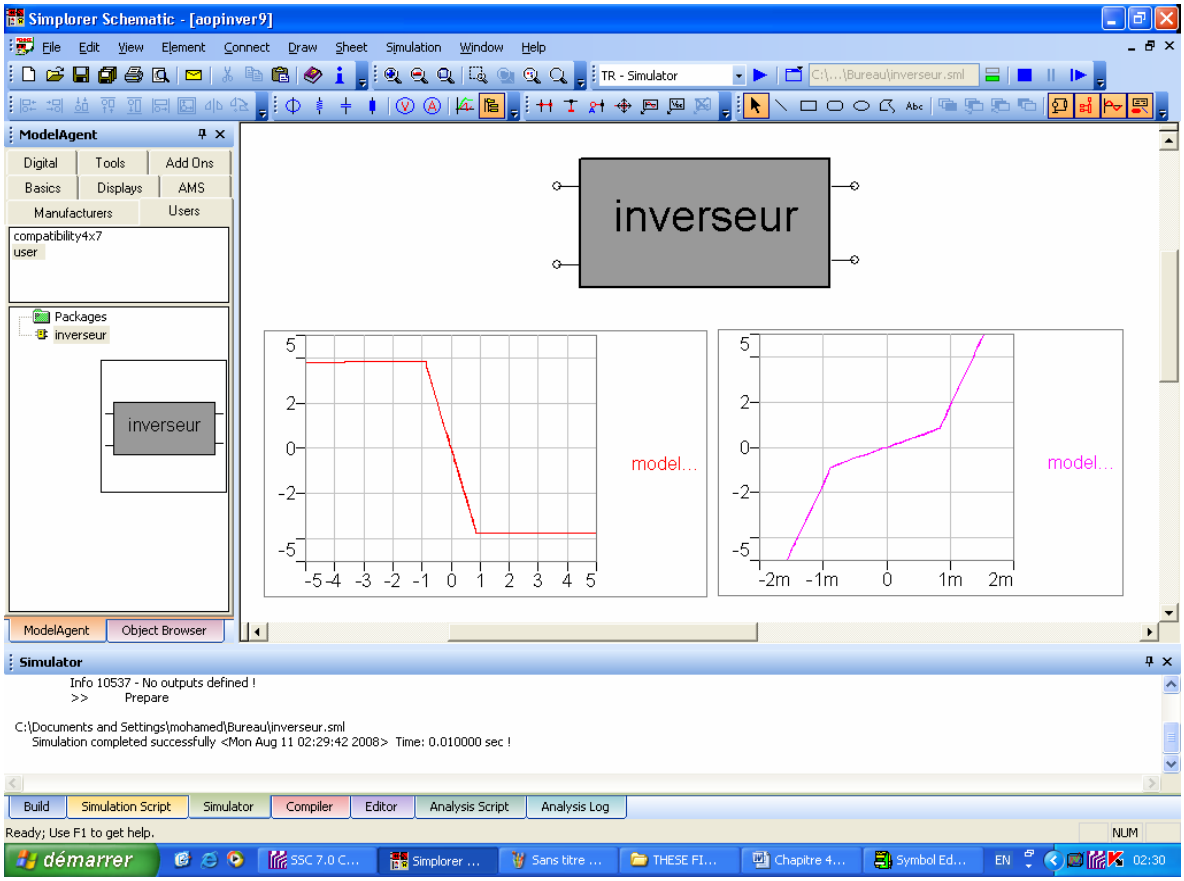


Figure IV.31 : Implémentation du modèle de l'amplificateur inverseur dans le simulateur Simplorer.

Les résultats obtenus avec VHDL-AMS sont tout à fait conformes à ceux obtenus avec PSPICE, il a été par contre possible avec le VHDL-AMS de simuler entièrement le système, ce qui est impossible avec PSPICE.

De plus, le temps de simulation avec le VHDL-AMS est inférieur d'un facteur de 80 fois à celui de la simulation SPICE de l'amplificateur inverseur (figure IV.32).

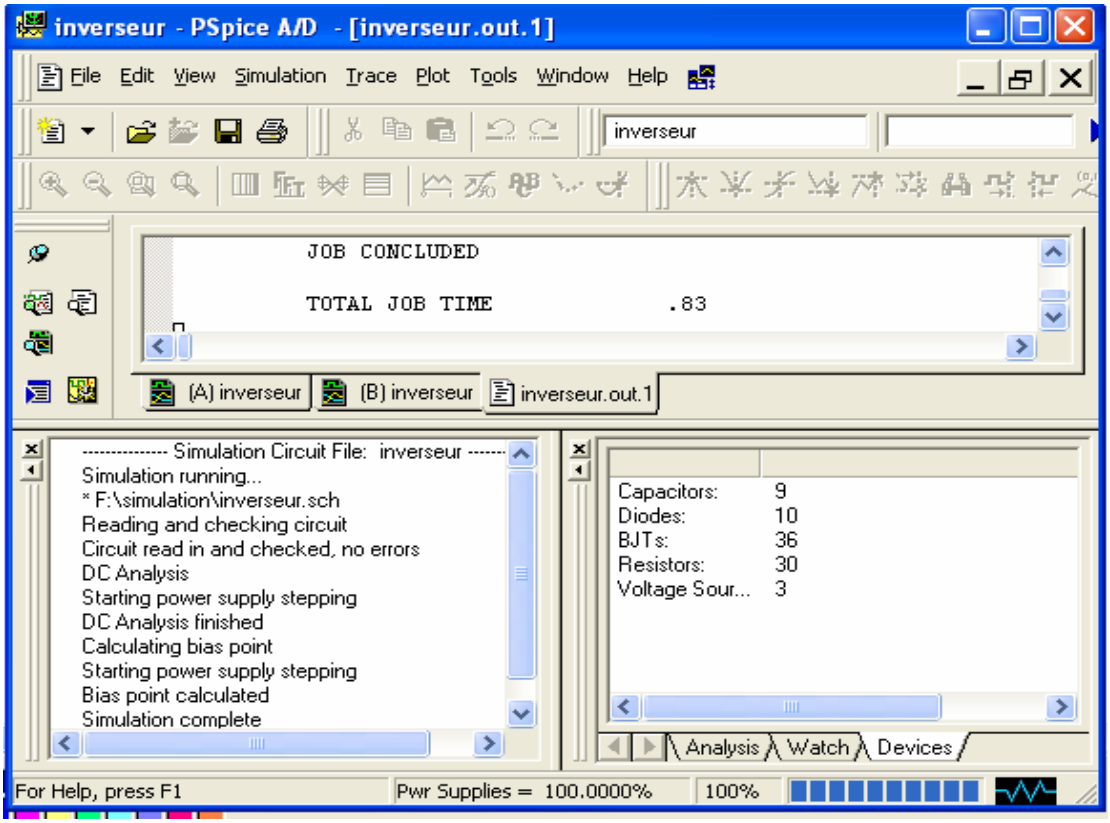


Figure IV.32 : temps de simulation de l'inverseur avec Pspice.

IV.5.Conclusion :

Ce chapitre a tenté de mettre en évidence les points clés de l'écriture de modèles comportementaux à l'aide soit d'un langage comportemental, pouvant être adapté à la description de systèmes complexes de haut-niveau selon la technique de piece-wice linear (P.W.L), dédiée à l'analogique, soit d'un langage structurel de type SPICE. 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 ce chapitre dans le cadre du développement modèle comportemental VHDL-AMS d'un amplificateur inverseur.

L'écriture d'un modèle en VHDL-AMS est directe et la difficulté des équations est transparente pour le programmateur. En effet, l'écriture d'une équation physique complexe – équation différentielle non linéaire nécessite une instruction simultanée, soit une

ligne de code en VHDLAMS. Puis la résolution est assurée par les puissants simulateurs VHDL de type Simplorer développés pour la CAO électronique.

Deux avantages supplémentaires du VHDL-AMS servent les intérêts des industriels :

- le langage est normalisé (IEEE 1076.1 199). Les modèles sont donc indépendants des outils de simulation et portables. Ceci facilite leur ré-utilisation.
- le langage VHDL-AMS permet la multi-abstraction des modèles : des modèles plus ou moins détaillés (fonctionnels, comportementaux, ou physiques) peuvent être décrits et simulés. Ceci facilite la conception hiérarchique des systèmes complexes.

Les concepteurs de circuits électroniques, sont donc très intéressés par la modélisation en langage
VHDL-AMS.

CONCLUSION

GENERALE

Dans ce travail nous avons ciblé certains objectifs concernant la réduction de la taille mémoire et la minimisation du temps de simulation dont souffrent généralement les circuits électriques à cause de leur complexité . Nous avons alors présenté de nouvelles approches prenant en compte les deux paramètres cités ci-dessus à travers des techniques qui ont fait usage dans des travaux de recherches antérieurs et dont certaines ont été développés à travers cette thèse.

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. [3]

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 modélisation comportementale VHDL est une autre clé de la conception système actuelle. Il apparaît primordial de pouvoir se baser sur des modèles fonctionnels, exécutables, et échangeables pour vérifier, par simulation tout au long du processus de conception, la conformité au cahier des charges, mais aussi pour optimiser les performances du système et ce, avant d'entamer les démarches de matérialisation et de réalisation technologique. Une clé de ces évolutions est la normalisation du langage de description de matériel VHDL-AMS. La modélisation multi abstractions et multi domaines avec le VHDL-AMS semble, en effet, être particulièrement prometteuse pour les systèmes électroniques complexes.

Dans ce mémoire, La tendance est d'aller au bout de cette logique de modélisation et d'avoir une approche uniquement basée sur les modèles. Ceux-ci sont une représentation d'un objet pouvant être proposée à un niveau d'abstraction plus élevé. Le modèle de haut niveau est nécessaire pour préparer une simulation VHDL. Il est particulièrement adapté pour avoir de bonnes performances de vitesse de simulation.

Dans un premier temps, un modèle fonctionnel de l'amplificateur inverseur a été développé ; il a permis de vérifier le bon fonctionnement du circuit réel. Puis, dans une seconde étape, le modèle fonctionnel a été remplacé par un modèle comportemental VHDL-AMS. Les résultats de la simulation comportementale obtenus vérifient ceux obtenus pratiquement. Le gain en temps de simulation est relativement important. Par conséquent, l'optimisation des paramètres est beaucoup plus facile en utilisant le modèle comportemental.

Le présent travail a tenté de mettre en valeur les avantages qu'apporte la modélisation comportementale VHDL-AMS tant au niveau de la simulation analogique et mixte, qu'au niveau des méthodologies de conception hiérarchique.

De manière globale, nous pensons qu'il faut expérimenter encore davantage avec des simulateurs VHDL sur des exemples de plus en plus complexes pour mieux définir les problématiques de la simulation des grands systèmes complexes. Les développements de modèles VHDL-AMS sont de plus en plus nombreux, et l'on voit de plus en plus d'industriels lancer des projets concrets, conscients des bénéfices qu'apporte la conception préalable d'un modèle prototype. Les contraintes matérielles en terme de temps de simulation et de ressource CPU limitent encore à l'heure actuelle la finesse globale de la modélisation d'un système complexe complet que l'on peut simuler. La démarche la plus riche se situe sûrement dans la simulation des systèmes à partir de modèles de différents niveaux.

ANNEXE 1

Paquetages utilitaires

Lors de l'évolution du langage VHDL vers le standard IEEE 1164, un package a été ajouté au langage VHDL initial. Ce package est constitué d'un ensemble de fonctions et de déclarations de types que l'on peut utiliser et appeler au sein de descriptions.

Nous allons considérer les versions préliminaires de ces paquetages et plus précisément:

• Le paquetage ENERGY_SYSTEMS qui définit un certain nombre de constantes utiles pour la description de systèmes à conservation d'énergie.

• Le paquetage ELECTRICAL_SYSTEMS qui définit la nature electrical et des déclarations relatives.

On suppose que ces paquetages sont compilés dans la bibliothèque IEEE_PROPOSED. Il faut donc déclarer les clauses suivantes pour pouvoir utiliser les déclarations de ces paquetages dans un modèle:

```
library ieee_proposed;
use ieee_proposed.energy_systems.all;
use ieee_proposed.electrical_systems.all;
```

1. Paquetage ENERGY_SYSTEMS.

```
library ieee;
use ieee.math_real.all;
package ENERGY_SYSTEMS is
-- subtype declarations
subtype ENERGY is REAL tolerance "DEFAULT_ENERGY";
subtype POWER is REAL tolerance "DEFAULT_POWER";
subtype PERIODICITY is REAL tolerance "DEFAULT_PERIODICITY";
-- common scaling factors
constant ATTO : REAL := 1.0e-18;
```

```
constant FEMTO : REAL := 1.0e-15;
constant PICO : REAL := 1.0e-12;
constant NANO : REAL := 1.0e-9;
constant MICRO : REAL := 1.0e-6;
constant MILLI : REAL := 1.0e-3;
constant CENTI : REAL := 1.0e-2;
constant DENCI : REAL := 1.0e-1;
constant DEKA : REAL := 1.0e+1;
constant HECTO : REAL := 1.0e+2;
constant KILO : REAL := 1.0e+3;
constant MEGA : REAL := 1.0e+6;
constant GIGA : REAL := 1.0e+9;
constant TERA : REAL := 1.0e+12;
-- permittivity of vacuum <FARADS/METER>
constant EPS0 : REAL := 8.8542e-12;
-- permeability of vacuum <HENRIES/METER>
constant MU0 : REAL := 4.0e-6 * MATH_PI;
-- electron charge <COULOMB>
constant Q : REAL := 1.60218e-19;
-- Boltzmann constant <JOULES/KELVIN>
constant K : REAL := 1.38066e-23;
-- Relative permittivity of silicon
constant EPS_SI : REAL := 11.8;
-- Relative permittivity of silicon dioxide
constant EPS_SIO2 : REAL := 3.9;
-- Young's Modulus for silicon <PASCALS>
constant E_SI : REAL := 190.0e+9;
-- Young's Modulus for silicon dioxide <PASCALS>
constant E_SIO2 : REAL := 73.0e+9;
-- Poisson's Ratio for silicon <100> orientation
constant NU_SI : REAL := 0.28;
-- Acceleration due to gravity <METERS/SECOND_SQRD>
constant GRAV : REAL := 9.81;
-- attribute declarations
```

```
constant SYMBOL : STRING
end package ENERGY_SYSTEMS;
```

2. Paquetage ELECTRICAL_SYSTEMS.

```
library ieee;
use ieee.energy_systems.all;
package ELECTRICAL_SYSTEMS is
-- subtype declarations
subtype VOLTAGE is REAL tolerance "DEFAULT_VOLTAGE";
subtype CURRENT is REAL tolerance "DEFAULT_CURRENT";
subtype CHARGE is REAL tolerance "DEFAULT_CHARGE";
subtype RESISTANCE is REAL tolerance "DEFAULT_RESISTANCE";
subtype CAPACITANCE is REAL tolerance "DEFAULT_CAPACITANCE" ;
subtype MMF is REAL tolerance "DEFAULT_MMF";
subtype FLUX is REAL tolerance "DEFAULT_FLUX";
subtype INDUCTANCE is REAL tolerance "DEFAULT_INDUCTANCE";
-- nature declarations
nature ELECTRICAL is
VOLTAGE across
CURRENT through
ELECTRICAL_REF reference;
nature ELECTRICAL_VECTOR is array (NATURAL range <>) of ELECTRICAL;
nature MAGNETIC is
MMF across
FLUX through
MAGNETIC_REF reference;
nature MAGNETIC_VECTOR is array (NATURAL range <>) of MAGNETIC;
-- attribute declarations
attribute SYMBOL of VOLTAGE: subtype is "Volt";
attribute SYMBOL of CURRENT: subtype is "Ampere";
attribute SYMBOL of CHARGE: subtype is "Coulomb";
attribute SYMBOL of RESISTANCE: subtype is "Ohm";
attribute SYMBOL of CAPACITANCE: subtype is "Farad";
```

attribute SYMBOL **of** MM: **subtype is** "Ampere-Turns";

attribute SYMBOL **of** FLUX: **subtype is** "Weber";

attribute SYMBOL **of** INDUCTANCE: **subtype is** "Henry";

end package ELECTRICAL_SYSTEMS;

BIBLIOGRAPHIE

[1] Sébastien SNAIDERO,
" Modélisation multidisciplinaire VHDL-AMS de systèmes complexes : vers le Prototypage Virtuel ",
thèse de Doctorat, Université Louis Pasteur Strasbourg I, décembre 2004.

[2] Yannick Hervé,
" VHDL-AMS : Applications et enjeux industriels ",
Dunod, Paris, 2002.

[3] F. Lémery,
" Modélisation comportementale des circuits analogiques et mixtes ",
thèse de Doctorat, institut polytechnique de Grenoble, Décembre 1995.

[4] N. Bourouba,
Développement des techniques de test pour des circuits analogiques integrés de types S.S.I. ,
thèse de Doctorat,Université Ferhat Abbas Setif, Nouvembre 2007.

[5] Patricia Desgreys,
" Modélisation Physique en VHDL-AMS ",
l'Ecole Supérieure des Télécommunications de Paris, 2002.

[6] Christophe PAOLI,
" Validation de descriptions VHDL fondée sur des techniques issues du domaine du test de logiciels ",
Université de CORSE – PASQUALE PAOLI, décembre 2001.

[7] T. T. Lang,
" circuits fondamentaux de l'électronique analogique ",
3e édition, technique et documentation, 1996.

[8] R. Perdriau,
"Méthodologie de prédiction des niveaux d'émission conduite dans les circuits intégrés, à l'aide de VHDL-AMS ",
thèse de Doctorat, université catholique de Louvain, laboratoire de microélectronique (DICE), Mars 2004.

[9] S. Djemmali,
" contribution à l'élaboration de méthodologies et D'outils d'aide à la conception de systèmes multi technologiques ",
Phd thesis, école nationale supérieur des communications, 2003.

[10] A. Vachoux,
" Modélisation des systèmes analogiques et mixtes, introduction à VHDL-AMS ",
Laboratoire de systèmes microélectroniques, version 2003.

[11] Juan-carlos Hamon,
" Méthodes et outils de la conception amont pour les systèmes et les microsystèmes », thèse de Doctorat, institut national polytechnique de Toulouse, Février 2005.

[12] Written by the staff of FERRANTI INTERDESIGN, INC.
" Functional blocks for monochip custom integrated circuits ",
1980.

[13] S. Mir,
" Conception et test intégré des dispositifs analogiques, mixtes et microsystèmes ",
thèse de Doctorat, institut national polytechnique de Grenoble, Mai 2005.

[14] A. Vachoux,
" Modélisation de Systèmes Intégrés Numériques Introduction à VHDL"
Laboratoire de systèmes microélectroniques STI-LSM , version 2003.

[15] M. Azizi,
" Covérification des systèmes intégrés " ,
thèse de Ph.D, Université de Montréal, Décembre 2000.

[16] S. Belkacem,
" Macro modélisation comportementale de circuits analogiques : application au circuit convoyeur de courant ",
thèse de Magister, université de Batna, 2005.

[17] **" IEEE- 488 Interface boards: User's manual ".**
Printed in the U.S.A, 2000.

[18] P.R. Wilson, Y. Kiliç , J.N. Ross, M. Zwolinski, Andrew D. Brown,
"Behavioural Modelling of Operational Amplifier Faults using VHDL-AMS ",
Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition, IEEE 2002.

[19] Catherine BAYOL,
" Une approche structurelle et comportementale de modélisation pour la vérification de composants VLSI ",
thèse de Doctorat, Université Joseph Fourier, Grenoble1, décembre 1995.

[20] Peter j. Asbenden,
" The Designer's Guide to VHDL ",
Morgan Kaufmann Publishers, Inc. San Francisco, California, 2000.

[21] **Principes de la simulation électrique**

http://www.comelec.enst.fr/~porte/oceane/doc/documents/envsimu/principe/principe.html

[22] Thierry LAGUTERE,
"Conceptions et modélisations d'oscillateurs et de leurs boucles à verrouillage de Phase associées pour des applications de radiocommunications mobiles professionnelles ",
thèse de doctorat,Ecole superieure d'ingénieurs de Poitiers, Février 2005.

[23] François BERRY,
" Cahiers de Microélectronique , Vol. I ",
DESS Micro-électronique analogique et microtechnologies, 2000-2001.

[24] François BERRY,
" Cahiers de Microélectronique , Vol. III ",
DESS Micro-électronique analogique et microtechnologies, 2002-2003.

[25] S. Graffi, G. Masetti, and D. Golzio,
"New Macromodels and Measurements for the Analysis of EM1 Effects in 741 Op-amp Circuits",
IEEE Transactions on electromagnetic compatibility, Vol. 33, N°.**1,** February 1991.

[26] Adam Jan. Morawiec,
" Amélioration des performances de la simulation des modèles décrits en langage de description de matériel " ,
thèse de Doctorat, Université Joseph Fourier, Grenoble1, octobre 2000.

[27] Emil Dumitrescu,
" Construction de modèles réduits et vérification symbolique de circuits industriels décrits au niveau RTL ",
thèse de Doctorat , Université Joseph Fourier, Grenoble1, octobre 2003.

[28] David GUIHAL,
" Modélisation en langage VHDL-AMS des systèmes pluridisciplinaires ",
Thèse de doctorat, Université Toulouse III, Mai 2007.

[29] Russell Kao,
" Piecewise Linear Models for Switch-Level Simulation ",
Technical Report: CSL-TR-92-532, Stanford University, June 1992.

[30] M. Rewien´ski and J. White,
"A Trajectory Piecewise-Linear Approach to Model Order Reduction and Fast Simulation of Nonlinear Circuits and Micromachined Devices ",
IEEE Transactions on Computer –Aided Design of Integrated Circuits and Systems, VOL. 22, NO. 2, February 2003.

[31] P. Poure, R. Kadri, L. Hébrard et F. Braun,
" Méthodologie d'intégration de commandes numériques pour dispositifs d'électronique de puissance basée sur l'utilisation du langage VHDL-AMS ",
Journal sur l'enseignement des sciences et technologies de l'information et des systèmes, Volume 2, Laboratoire d'Électronique et de Physique des Systèmes Instrumentaux (LEPSI), Université Louis Pasteur, IN2P3/CNRS, 2003.

[32] Youssef KEBBATI,
" Développement d'une Méthodologie de Conception Matériel à Base de Modules Génériques VHDL/VHDL-AMS en Vue d'une Intégration de Systèmes de Commande Electriques ",
Thèse de doctorat, Université Louis Pasteur – Strasbourg I, Décembre 2002.

[33] Thierry Schneider,
" VHDL : méthodologie de design et techniqus avancées ",
Série Principes électroniques, Dunod, Paris, 2001.

[34] Jaques Weber, Maurice Meaudre,
" Le langage VHDL, cours et exercices ",
2eme édition, Dunod, Paris, 2001.

[35] Michel Aumiaux,
" Initiation au langage VHDL ",
2eme édition, Dunod, Paris, 1999.

[36] C. TURCHETTI and G. MASETTI.
"A macromodel for integrated all MOS operational amplifiers",
IEEE Journal of Solid-State Circuit, vol. 18(No. 4):pp. 389-394, August 1983.

[37] Yannick Hervé,
" VHDL-AMS : Un langage pour l'électronique du futur ",
BEAMS, Laboratoire de Micro électronique, Université Bordeaux I, 2005.

Abstract

Study and application of a VHDL simulator to the implementation of models of electronic circuits

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.

The objective of this work is to implement functional models to be developed from a basic electronic circuit to the inverting amplifier in a standard VHDL simulator, while providing a limited memory space and time simulation reduced taking into account accuracy. The production of these models was made by using the technique of Piece-wise linear (PWL) on the transfer characteristics, entry and exit. This technique provides an electric model which is simplified deputy linear equations describing the function of the circuit.

Key words: VHDL-AMS, VHDL simulator, behavioral modeling, simulation, electric characteristics.