

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE BATNA

MEMOIRE

Présenté à

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

Pour l'obtention du diplôme de

MAGISTER
MICROELECTRONIQUE
Option: IC Design

Par
BENACER Imad

Thème

***MODELISATION COMPORTEMENTALE
DE SPICE A VHDL-AMS***

Devant le jury constitué de :

Pr. OUNISSI Abdelhamid	Prof. Université de Batna	Président
Dr. DIBI Zohir	M.C. Université de Batna	Rapporteur
Pr. CHAABI Abdelhafid	Prof. Université de Constantine	Examineur
Pr. MAHAMDI Ramadane	Prof. Université de Batna	Examineur
Dr. DJEFFAL Fayçal	M.C Université de Batna	Examineur

Je dédie ce travail à :

Ma mère,

Mon père,

Mes sœurs,

Tous mes amis sans exception.

Remerciements

Tout d'abord, je Remercie le bon Dieu le tout puissant pour la bonne santé, la volonté et la patience qu'il m'a donné pour accomplir ce travail.

Ce travail a été effectué au laboratoire d'Electronique Avancée de l'université de Batna. J'adresse mes remerciements les plus sincères au Docteur Zohir DIBI, Chef de département d'Electronique de l'université de Batna, 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.

Nous tenons à remercier également les membres de jury Pr. OUNISSI Abdelhamid, Pr. CHAABI Abdelhafid, Pr. MAHAMDI Ramadane, Dr. DJEFFAL Fayçal de Bien vouloir accepter d'examiner et de juger ce modeste travail.

Je remercie également tous ceux qui m'ont accompagné ou que j'ai croisés durant ces deux années au sein du laboratoire(LEA), tous ceux qui m'ont soutenu, encouragé et donné l'envie de mener à terme ce travail.

Enfin, je dois une dette certaine à ma famille et à mes parents, qui ont été mes plus fidèles supporteurs et qui m'ont aidé à traverser cette période.

Benacer Imad

SOMMAIRE

Introduction Générale	1
-----------------------------	---

Chapitre I

Techniques de modélisation des circuits

I.1 Introduction	4
I.2 Modèle	4
I.3 Analyse et synthèse	4
I.4 Types de comportements	5
I.5 Types de formalismes	7
I.6 Macro-modèle	9
I.7 Simulation électrique	10
I.7.1 Techniques de simulation	11
I.7.1.1 Simulation logique	12
I.7.1.2 Simulation analogique	12
I.7.1.3 Simulation mixte logique-analogique	14
I.8 Modélisation	16
I.8.1 Objectif de la modélisation	17
I.8.2 Méthodologie de modélisation des circuits analogiques	17
I.8.3 Exemples analogiques et digitaux	17
I.8.4 Modèles et primitives	18
I.8.5 Techniques de modélisation	18
I.9 Conclusion	23

Chapitre II

Langage de modélisation et simulateur

II.1 Introduction	24
II.2 Choix du langage de modélisation	24
II.2.1 Langage de programmation orientée objet	24
II.2.2 Langage de modélisation numérique	24
II.2.3 Langage mathématique formel explicite	24
II.2.4 Langage de modélisation implicite dédiée à l'électronique	25
II.2.5 Langage de modélisation analogique multi-domaines	25
II.2.6 Langage de modélisation mixte multi-domaines	25
II.3 Méthodes de modélisation VHDL-AMS	26
II.4 ORCAD PSPICE	27

II.4.1 Définition	27
II.4.2 Limitations de SPICE.....	28
II.5 VHDL-AMS.....	29
II.5.1 Histoire de VHDL-AMS	29
II.5.2 Avantage de VHDL-AMS.....	30
II.5.3 Limites de VHDL-AMS.....	30
II.5.4 Evolution liée à VHDL-AMS	31
II.6 Choix du logiciel de simulation	32
II.6.1 Simulateurs de première génération à code apparent.....	32
II.6.2 Possibilités offertes par l'interfaçage avec VHDL-AMS	33
II.6.3 Nouveaux simulateurs disposant d'un GUI (Graphical User Interface)	33
II.7 Simplorer.....	34
II.7.1 Introduction	34
II.7.2 Interface graphique.....	35
II.8 Modèle VHDL-AMS	36
II.8.1 Structure d'un modèle VHDL-AMS	36
II.8.2 Classes d'objets	39
II.8.3 Déclarations simultanées.....	40
II.9 Apports de VHDL-AMS	43
II.10 Application à la modélisation hiérarchique.....	45
II.11 Conclusion.....	46

Chapitre III

Simulation des circuits électroniques

III.1 Introduction.....	47
III.2 Connexion Analogique	47
III.3 Connexion Physique	47
III.3.1 Resistance	48
III.3.2 Capacité	49
III.3.3 Inductance	50
III.3.4 Diode.....	55
III.4 Connexion Mathématique.....	56
III.5 Générateurs	58
III.5.1 Sources indépendantes constantes	58
III.5.2 Source de tension sinusoïdal.....	59
III.5.3 Générateur d'impulsion	60

III.6 Trigger de Schmitt	62
III.6.1 Trigger Inverseur	62
III.6.1.1 Modèle Pspice du Trigger Inverseur.....	63
III.6.1.2 Modèle VHDL-AMS du Trigger Inverseur	65
III.6.2 Trigger non inverseur.....	66
III.6.2.1 Modèle pspice du trigger non inverseur	68
III.6.2.2 Modèle VHDL-AMS de trigger non inverseur.....	69
III.7 Contrôle du système hybride	70
III.7.1 Modélisation par pspice	70
III.7.2 Modélisation par VHDL-AMS	71
III.8 Machine à courant continu.....	72
III.8.1 Modèle électrique	73
III.8.2 Modélisation par pspice	73
III.8.3 Modélisation par VHDL-AMS	75
III.9 Conclusion	76

Chapitre IV

Boucle à verrouillage de phase (PLL)

IV.1 Introduction	77
IV.2 Détection d'évènements analogiques	77
IV.3 Principe de fonctionnement.....	78
IV.4 Modélisation d'une PLL.....	81
IV.4.1 Modèle VHDL-AMS de la PLL	82
IV.4.1.1 Comparateur de phase	82
IV.4.1.2 Filtre de boucle	83
IV.4.1.3 Oscillateur commandé en tension.....	83
IV.4.2 Modèle Pspice de la PLL.....	85
IV.5 Conclusion	87
Conclusion Générale.....	88
Glossaire	90
Bibliographie	91

Introduction Générale



Introduction générale

La micro-électronique consiste en la réalisation miniaturisée de fonctions électroniques de plus en plus complexes sur un seul support "System On Chip" (SOC) et des ASICs (Application Specific Integrated Circuit) mixtes. Au départ, le but de la micro-électronique était la réduction du poids et du volume des appareils, mais ces deux critères sont devenus secondaires face à l'amélioration de la fiabilité et la réduction du prix de revient (Time_to_market) que permet l'intégration.

Les concepteurs de systèmes ont donc un grand besoin d'outils permettant un développement rapide et à moindre coût. Cette attente devrait trouver une réponse dans le "Prototypage Virtuel" (PV) des systèmes, en utilisant des langages de description de matériel (Hardware Description Language – HDL), qui permettrait de réduire à leurs minimal les phases de test sur des prototypes physiques, très coûteux en temps et en argent.

Les simulateurs de la famille SPICE proposent de nombreuses primitives (résistances, condensateurs, bobines, transistors MOS et Bipolaires 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'autres, 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. On peut ainsi conclure que lorsque le nombre de transistors devient important (quelques centaines à quelques milliers en conception analogique), le temps de simulation devient élevé et des problèmes de convergence peuvent apparaître rendant plus complexe la simulation d'un circuit analogique.

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.

On peut faire la distinction entre deux techniques utilisées pour réaliser des modèles comportementaux de fonctions analogiques : la macromodélisation et l'utilisation d'un langage de modélisation comportementale.

La macromodélisation consiste à décrire le comportement d'un circuit par l'utilisation des primitives d'un simulateur. Ces primitives sont par exemples des diodes, des éléments passifs, mais aussi et surtout des sources de tension (ou courant) contrôlées auxquelles on peut associer des équations liant les grandeurs du circuit. Les modèles obtenus par cette approche permettent de réduire considérablement les temps de simulations [1].

Les langages de modélisation comportementale permettent de développer des modèles de systèmes électriques mixtes (analogique et numérique) mais aussi de systèmes électriques et non-électriques (mécanique, hydraulique, thermique...) par l'utilisation d'une description textuelle. Ces langages sont de plus en plus utilisés. Parmi eux, on citera le langage MAST du simulateur SABER, le langage verilog-AMS, ainsi que le VHDL-AMS [1].

Nous nous proposons donc de comparer les deux approches afin de montrer qu'à partir d'une méthodologie de modélisation donnée, il est facile de développer un modèle comportemental aussi bien avec un langage comportemental (VHDL-AMS) qu'avec un outil de macromodélisation (le module ABM d'ORCAD-PSPICE). L'intérêt est bien sûr ici de mettre en avant la méthode et non le langage ou le simulateur.

Organisation du mémoire

Les travaux présentés dans ce mémoire apportent une contribution à la macromodélisation comportementale utilisant PSPICE-ORCAD et la modélisation comportementale utilisant le langage VHDL-AMS pour différents circuits.

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

- *Le Chapitre I présente les techniques de Modélisation des Circuits,*
- *Le Chapitre II aborde les divers langages de modélisation ainsi que les différents simulateurs utilisés à cet effet,*
- *Le Chapitre III résume les simulations effectuées sur des circuits électroniques,*

- *Le Chapitre IV détaille la Boucle à verrouillage de phase (PLL) par les deux approches proposés et établit une comparaison.*

Une conclusion générale est donnée en fin de ce mémoire.

Chapitre I

Techniques de modélisation des circuits

I.1 Introduction

Ce chapitre présente un certain nombre de notions fondamentales: modèle, analyse, synthèse, représentations d'un modèle, formalismes de description de modèles, techniques de simulation, langage de description de matériel.

I.2 Modèle

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 (encore) connus.

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 tâche la plus difficile.

I.3 Analyse et synthèse

On considère ici deux types d'utilisation de modèles mathématiques: l'analyse et la synthèse.

Le processus **d'analyse** permet d'extraire ou de vérifier les propriétés d'un système. La **simulation** est un type d'analyse dynamique pour lequel le modèle est soumis à un ensemble de stimuli. Ceci génère un certain nombre de réponses qui permettent d'extraire des propriétés du modèle donc du système simulé. Le modèle doit ainsi avoir une **forme exécutable** utilisable par un **simulateur** (p. ex., mais pas exclusivement, un programme d'ordinateur).

La qualité des réponses obtenues d'un simulateur dépend fortement de la qualité des questions (les stimuli), de la qualité des modèles considérés et des techniques de simulation utilisées. Etant donné son caractère intrinsèquement abstrait, un modèle possède un certain domaine de validité qui délimite ses conditions d'utilisation. Au-delà de son domaine de validité, un modèle devient incorrect.

La **vérification** est un type d'analyse statique qui n'utilise pas de stimuli, mais qui examine un modèle par rapport à certaines propriétés qu'il devrait posséder.

Le processus de **synthèse** réalise un ensemble de transformations sur un modèle de manière à dériver un nouveau modèle plus détaillé et optimisé en fonction de contraintes de conceptions imposées (p. ex. minimisation de la surface, des délais ou de la consommation) [2].

I.4 Types de comportements

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.

Un **comportement continu** est mathématiquement représenté par une fonction à valeurs réelles d'une variable réelle indépendante, généralement le temps ou la fréquence (Figure I.1). Un tel comportement peut être représenté par des relations algébriques linéaires ou non linéaires, différentielles linéaires ou non linéaires.

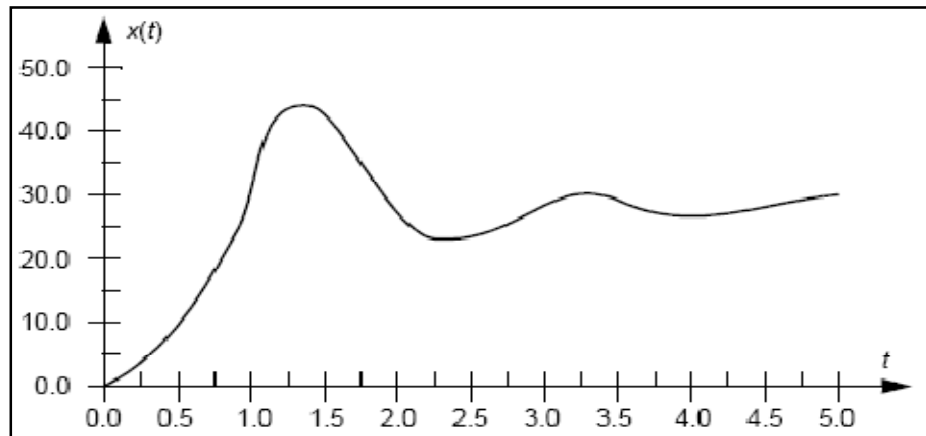


Figure I.1: Comportement continu [2].

On peut distinguer deux types de modèles à temps continu. Les modèles à **paramètres localisés** sont décrits par des équations différentielles ordinaires $\dot{x} = f(x, u, t)$, où x est la variable dépendante, u est un stimuli et t est le temps.

Exemples de comportements à temps continu à paramètres localisés:

$$u(t) = R \cdot i(t) \quad \text{algébrique linéaire} \quad (\text{I.1})$$

$$u(t) = C \cdot \frac{du}{dt} \quad \text{différentiel linéaire} \quad (\text{I.2})$$

$$i(t) = I_s (e^{u(t)/nU_T} - 1) \quad \text{algébrique non linéaire} \quad (\text{I.3})$$

$$u(t) = C(u(t)) \cdot \frac{du}{dt} \quad \text{différentiel non linéaire} \quad (\text{I.4})$$

Les modèles à **paramètres distribués** sont décrits par des équations aux dérivées partielles, comme par exemple l'équation de la diffusion:

$$\frac{\partial u}{\partial t} = \partial \cdot \frac{\partial^2 u}{\partial x^2} \quad (\text{I.5})$$

Un **comportement discret** est mathématiquement représenté par une équation aux différences. Les variables dépendantes sont à valeurs réelles, mais la variable indépendante est discrète (usuellement un multiple entier d'une unité de base). Un comportement discret peut être représenté comme

$$x_{k+1} = f(x_k, u_k, t_k) \text{ où } x_k = x(t_k), u_k = u(t_k) \text{ et } t_k = k \cdot t_0 \quad (\text{I.6})$$

t_0 représentant une période d'échantillonnage (Figure I.2).

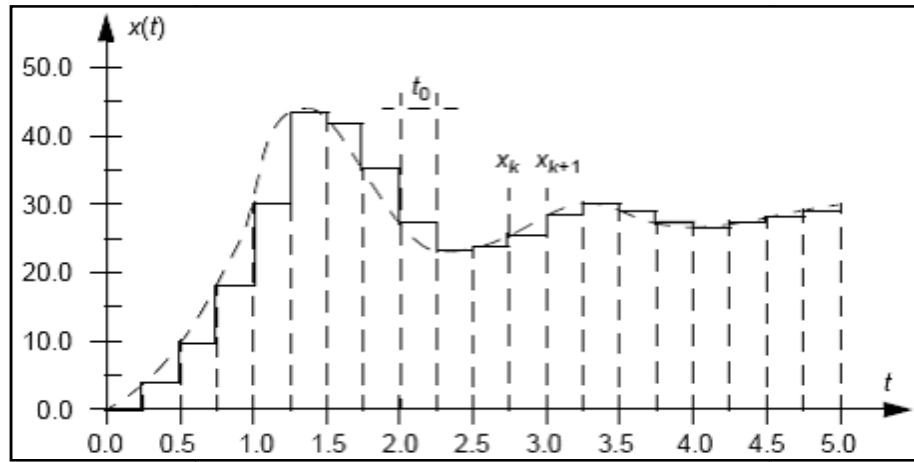


Figure I.2: Comportement discret [2].

Exemple: le comportement d'un filtre numérique à réponse impulsionnelle finie peut s'exprimer par la relation suivante:

$$y_{k+1} = a_0 x_k + a_1 x_{k-1} + a_2 x_{k-2} + \dots \quad (\text{I.7})$$

Où x_i représente un échantillon d'entrée, y_i représente un échantillon de sortie et a_i est un coefficient du filtre.

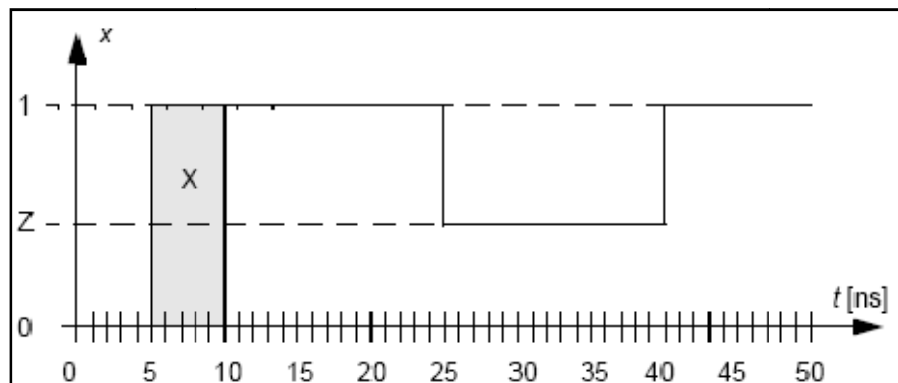


Figure I.3: Comportement logique [2].

On distingue les comportements logiques combinatoires dont les fonctions s'expriment par des équations booléennes sans délai, les comportements logiques asynchrones qui sont des comportements logiques combinatoires avec délais et les comportements logiques synchrones qui incorporent une fonction de mémorisation d'états séquencée par un signal d'horloge.

Les délais représentent la variable indépendante, le temps, et s'expriment comme des multiples entiers d'une unité de base appelée temps de résolution minimum.

Le comportement est dirigé par les événements, c'est-à-dire que l'état d'un système logique n'est défini qu'en fonction de changements d'états des variables d'entrée (signal d'horloge compris).

Exemple d'équations booléennes combinatoires:

$$s = a \oplus b \oplus c_i \quad (\text{I.8})$$

$$c_0 = (a \cdot b) + (a \cdot c_i) + (a \cdot c_i) \quad (\text{I.9})$$

Exemple d'équations booléennes synchrones (l'indice n dénote le pas temporel):

$$a^{(n)} = \overline{i \oplus i^{(n-1)}} \quad (\text{I.10})$$

$$b^{(n)} = \overline{i^{(n-1)} \oplus i^{(n-2)}} \quad (\text{I.11})$$

$$c = a \cdot b \quad (\text{I.12})$$

Dans l'exemple précédant, l'horloge est implicite. La variable b est par exemple égal à la variable i retardée de deux cycles.

I.5 Types de formalismes

On présente ici un certain nombre de formalismes qui peuvent être utilisés pour représenter des modèles de systèmes analogiques ou numériques.

I.5.1 Réseaux de Kirchhoff

La théorie des réseaux de Kirchhoff permet de définir des modèles de circuits électriques, et même des 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 nœud. Les bornes connectées au même nœud ont le même potentiel. La somme des courants pénétrant par les bornes connectées au même nœud 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 nœuds 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 I.4 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).

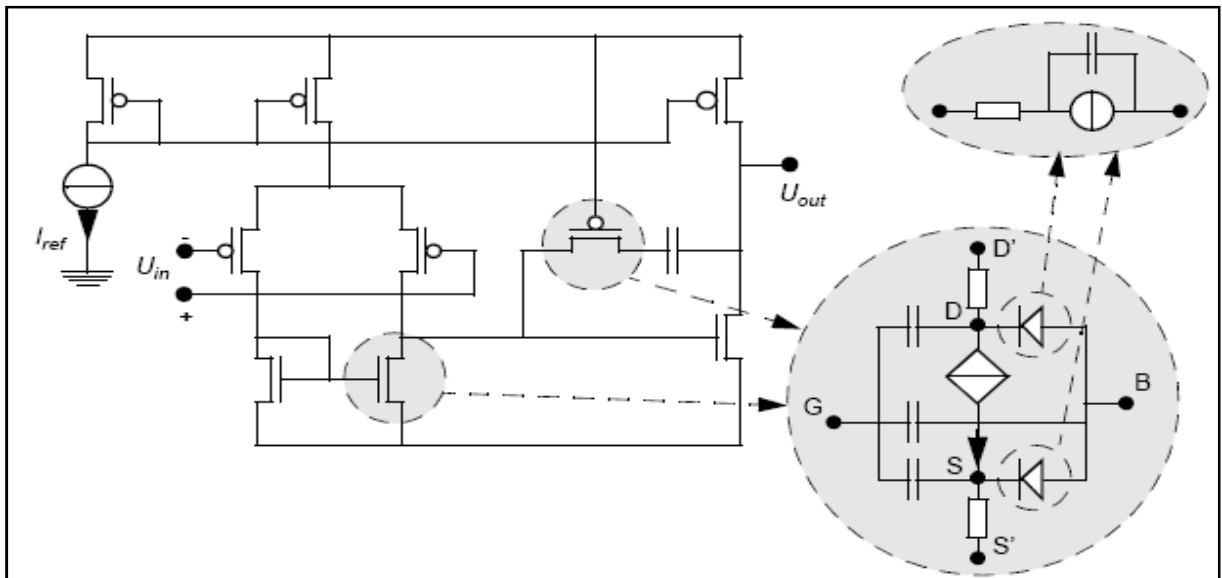


Figure I.4: Amplificateur CMOS représenté par un réseau de Kirchhoff [2].

I.5.2 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.

La Figure I.5 illustre un système de contrôle représenté par un diagramme de blocs. On distingue des blocs fonctionnels réalisant des fonctions de transfert (multiplicateur, dérivateur, intégrateur) et des nœuds réalisant des sommations/soustractions de signaux.

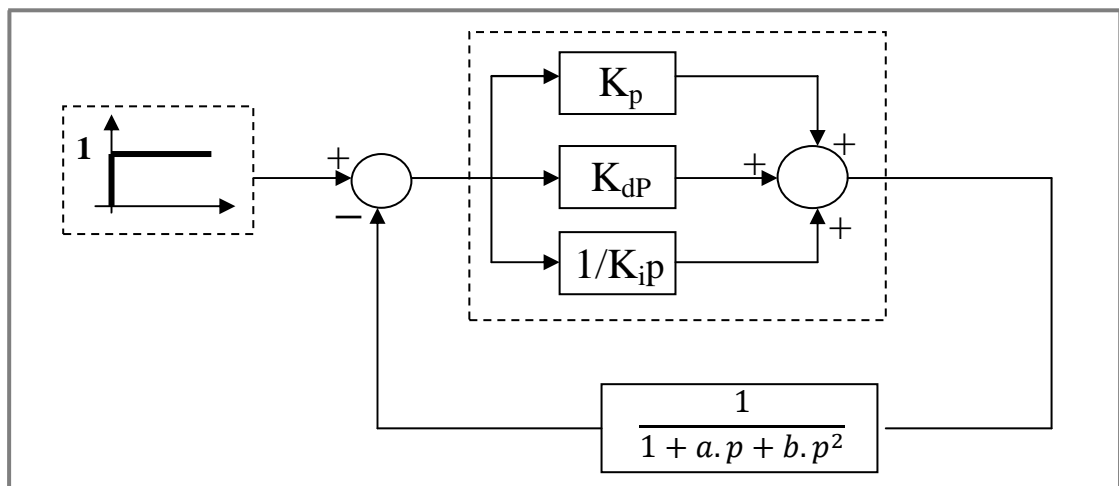


Figure I.5: Système de contrôle (régulateur PID) représenté par un diagramme de blocs.

I.6 Macro-modèle

Un macro-modèle est une représentation simplifiée, mais précise, d'un circuit ou d'un système à un niveau d'abstraction donné. Un macro-modèle est très souvent défini au niveau électrique, ou circuit. Un macro-modèle 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 macro-modèle est simulé plus rapidement que le modèle original. Il est aussi possible de définir un macro-modèle comportemental sous la forme d'un ensemble d'équations simplifiées.

Un exemple type de macro-modèle est le modèle de Boyle de l'amplificateur opérationnel (Figure I.6). Le modèle comprend trois étages. L'étage d'entrée a été dérivé par simplification de l'étage d'entrée du circuit réel (l'AO 741), d'où la paire différentielle. Les deux autres

étages ont été dérivés par des techniques constructives (*build-up*) utilisant des éléments simples comme des sources commandées. Le circuit réel comporte environ 25 transistors.

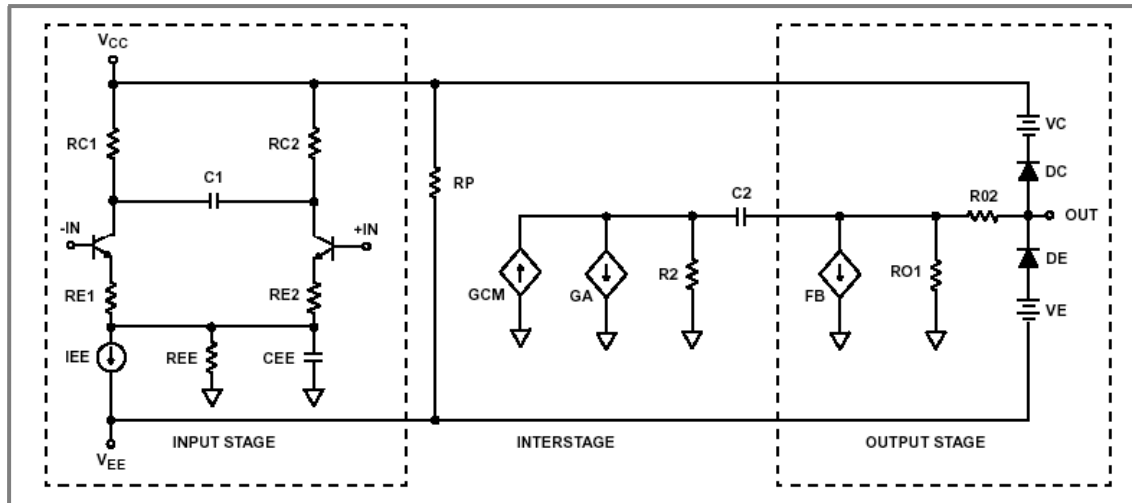


Figure 1.6: Macro-modèle de Boyle de l'amplificateur opérationnel [2].

Il n'est par contre pas toujours facile, ni même possible, de définir un macro-modèle suffisamment précis. Une raison majeure est qu'un macro-modèle est construit à partir d'un ensemble limité de primitives. Il existe toutefois des techniques numériques ou symboliques pour dériver un macro-modèle à partir d'un circuit complet.

I.7 Simulation électrique

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

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

- L'étude du **point de fonctionnement** du circuit ou analyse DC qui correspond à une étude en régime permanent,

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

I.7.1 Techniques de simulation

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.).

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

- Le moyen pour décrire le système à simuler (Langage de description),
- La description du système (Modèle),
- La description des entrées et sorties (E/S) du système (Test Bench),
- Le mécanisme de simulation du système qui a été conçu (Simulateur).

I.7.1.1 Simulation logique

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.7 donne l'algorithme général de la simulation dirigée par les événements (*event-driven simulation*).

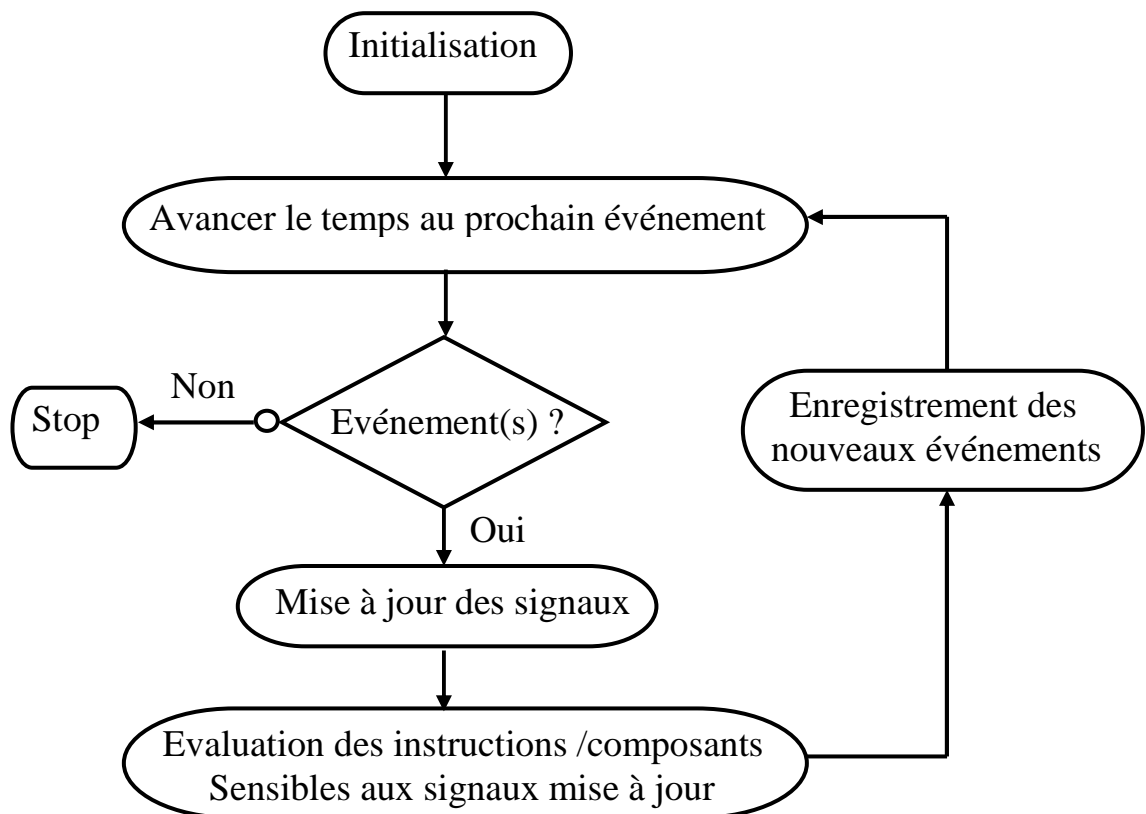


Figure I.7: Algorithme général de simulation dirigée par événements [2].

I.7.1.2 Simulation analogique

La simulation analogique est beaucoup plus complexe que la simulation logique et requiert ainsi plus de ressources (temps de calcul, mémoire). La simulation analogique implique la résolution d'équations différentielles et algébriques linéaires et non linéaires. Les solutions sont des tensions entre les nœuds 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 stimuli (sources et conditions initiales). La Figure 1.8 illustre un extrait de la réponse temporelle d'un circuit additionneur. Les stimuli sont définis comme une forme d'onde pulsée périodique avec des temps de montée et de descente non nuls (en haut). La réponse est un ensemble de formes d'ondes dont seuls un certain nombre de points sont effectivement calculés par la résolution des équations du circuit. Les formes d'ondes affichées (telle que celle du bas) sont obtenues par interpolation linéaire entre les points calculés. Il faut noter que l'intervalle entre les points calculés n'est pas constant pour des raisons de précision qui seront expliquées plus loin [2].

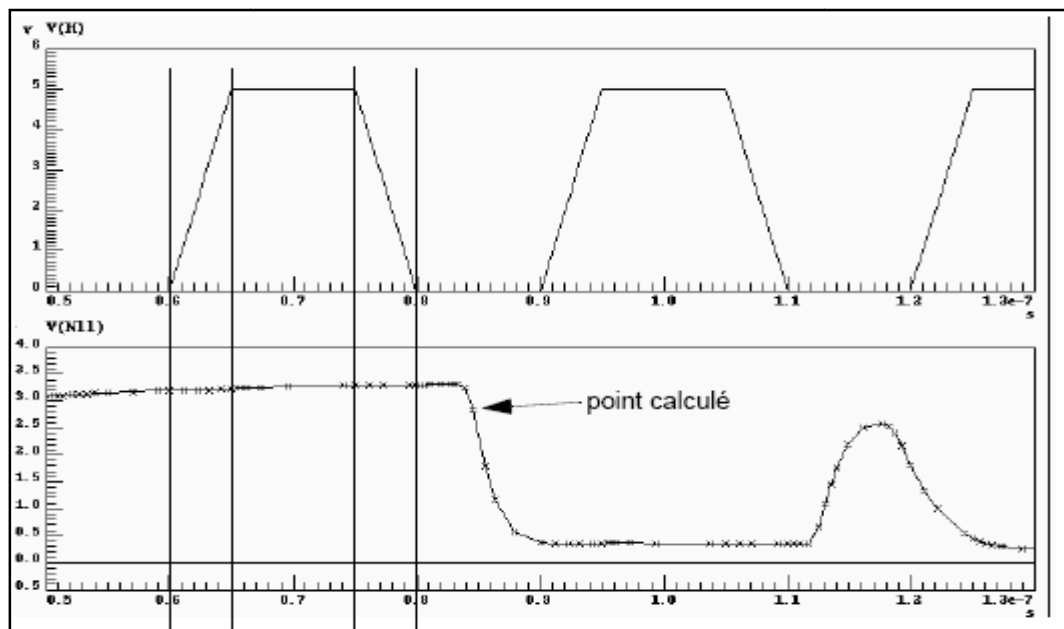


Figure 1.8: Réponse temporelle d'un additionneur (extrait) [2].

L'analyse DC (*direct current*) calcule l'état du circuit pour un ensemble de stimuli 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. La Figure I.9 illustre un résultat possible de l'analyse DC. Il s'agit de la caractéristique de transfert $i_d = f(u_{ds}, u_{gs})$ d'un transistor MOS obtenue par plusieurs analyses du point de repos pour des variations des tensions u_{ds} et u_{gs} .

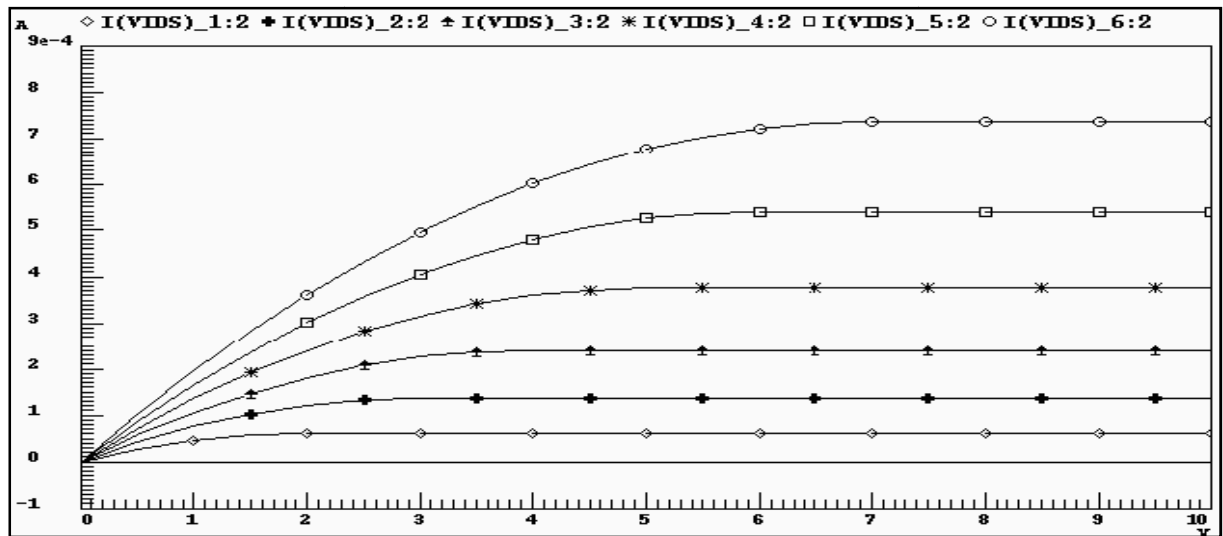


Figure I.9: Courbes de transfert DC $i_d = f(u_{ds}, u_{gs})$ d'un transistor MOS [2].

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) 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. La Figure I.10 illustre un résultat possible de l'analyse AC. Il s'agit du calcul du gain en dB et de la phase d'un amplificateur.

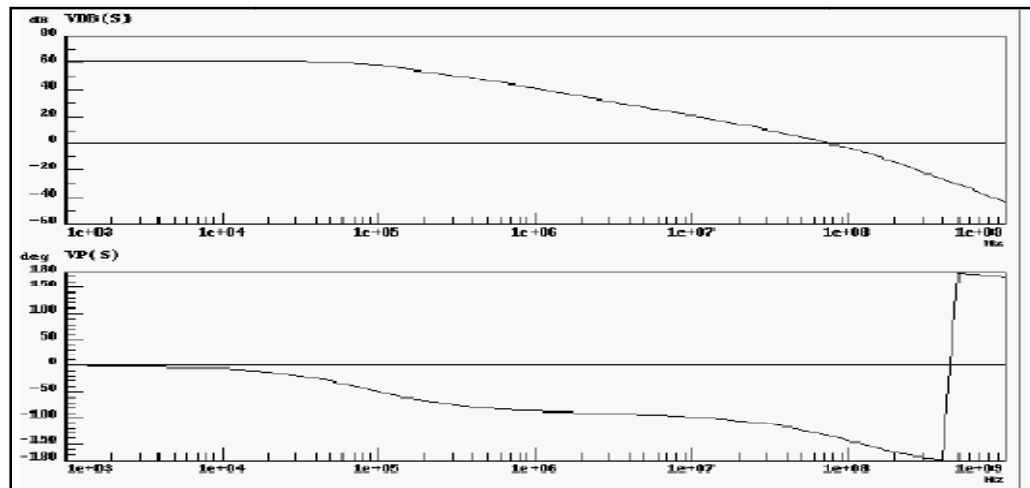


Figure I.10: Gain et phase d'un amplificateur [2].

I.7.1.3 Simulation mixte logique-analogique

La Table 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
Itérations à un temps donné	Affectation de signaux avec délais nuls (délai delta)	Résolution de systèmes non linéaires
Représentation du temps	Discret, multiple du MRT	Réelle
Gestion du temps	Dirigée par événements	Continue avec pas d'intégration variable
Contrôle du pas temporel	Evénements sur les signaux	Erreur de troncature locale ou équivalent
Types d'analyses	Temporelle	Temporelle, DC, AC

Table I.1: Récapitulation des caractéristiques de la simulation logique et analogique [2].

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

- 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).

La Figure I.11 présente un exemple d'interface analogique-logique capable de convertir une valeur analogique (tension ou courant) en une grandeur logique à trois états '0', '1' et 'X'. Le cycle d'hystérésis permet de prendre en compte du côté logique les transitions entre états stables [2].

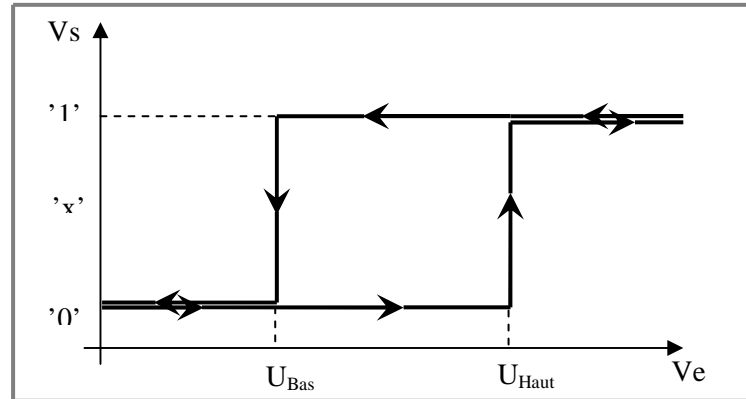


Figure I.11: Exemple d'interface analogique-logique.

I.8 Modélisation

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

Ainsi, lors de la conception de circuits intégrés tels qu'un amplificateur opérationnel ou une porte logique, des *modèles de composants* (transistors, diodes, résistances, capacités,...) sont utilisés par un simulateur électrique, dit analogique (le programme *SPICE* est sans doute le plus connu). Ces modèles décrivent les relations macroscopiques entre tensions et courants des diverses bornes, sous forme d'équations différentielles. Il s'agit donc d'une représentation mathématique de phénomènes physiques auxquels obéissent les composants.

Il existe deux approches permettant d'obtenir des modèles plus ou moins précis d'un composant ou d'une fonction électronique :

🔧 L'approche physique consiste à utiliser les modèles créés par les concepteurs de circuit intégrés et de modifier uniquement les paramètres. Cette approche peut être définie à partir de paramètres électriques (calculés à partir de mesures statiques, dynamiques, fréquentielles) ou à partir de paramètres technologiques caractéristiques du processus de fabrication et de la géométrie du composant.

🔧 L'approche comportementale consiste à modéliser un composant ou un circuit par l'évolution de ses entrées/sorties en réponse à différents stimuli.

I.8.1 Objectif de la modélisation

Elle a pour but de caractériser par une fonction mathématique ou un modèle numérique les différents composants qui constituent le circuit. C'est la partie la plus délicate du processus puisque des modèles simplifiés diminuent la précision, tandis que des modèles élaborés consomment beaucoup en mémoire et en temps de calcul [3].

I.8.2 Méthodologie de modélisation des circuits analogiques

Pour des raisons vues précédemment, la modélisation structurelle ne semble pas être adaptée à la simulation de systèmes analogiques complexes. Pour pouvoir simuler et synthétiser de tels systèmes, il est nécessaire de créer des bibliothèques de modèles comportementaux des fonctions élémentaires afin de réduire le temps de calcul.

Les modèles seront d'abord réalisés dans le cas du composant idéal, puis nous introduisons progressivement les influences de chaque paramètre sur le comportement du composant ou du circuit.

L'objectif est de modéliser les fonctions analogiques élémentaires et de les remplacer par leur modèle représentatif dans des simulateurs globaux. Créer une bibliothèque de fonction de base permettrait au concepteur de choisir, de manière rapide et précise, les modèles des différentes fonctions qui composent son circuit.

Il serait possible de concevoir des modèles très complets utilisables dans n'importe quelle zone de fonctionnement. Cependant, ces modèles très difficiles à mettre en œuvre consomment beaucoup de temps de calcul et ne représentent pas l'application de manière précise [4].

I.8.3 Exemples analogiques et digitaux

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

Dans le domaine analogique, les signaux restent des grandeurs électriques, fonctions continues du temps (ou de la fréquence). Les modèles comportementaux analogiques peuvent être représentés par un ensemble simplifié d'équations différentielles, des fonctions mathématiques non-linéaires ou linéaires par morceaux ou des tables de données. Notons que

le terme de *macromodélisation* est couramment employé en analogique. Il désigne en fait une méthode particulière de modélisation comportementale qui consiste en une simplification du schéma et, essentiellement, en l'utilisation de sources idéales contrôlées, proposées par les simulateurs de type SPICE, pour exprimer des relations entre tensions et courants. Cette approche est cependant assez limitée aux primitives du simulateur, introduit des effets parasites et peut poser des difficultés de convergence [4].

I.8.4 Modèles et primitives

➤ **Le modèle**

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 tâche la plus difficile.

➤ **Les primitives**

Tous les simulateurs analogiques proposent aux utilisateurs un jeu de composants de base permettant de décrire la topologie d'un circuit électrique. Ces éléments de base sont appelés les primitives du simulateur. Ces composants sont entre autre, les éléments actifs comme les diodes et les transistors (bipolaire, MOS, JFET), les éléments passifs comme les résistances, les capacités, les inductances, et les sources de courant et de tension (Gvalue, Evalue...); ce sont des éléments idéaux qui permettent d'exprimer facilement des relations mathématiques entre tensions et courants dans le domaine fréquentiel ou temporel.

Cependant, chaque simulateur possède sa propre syntaxe, ce qui freine l'utilisation de ces fonctions si le but est l'échange des macro-modèles [3].

I.8.5 Techniques de modélisation

Nous pouvons distinguer principalement trois techniques de modélisation qui sont :

I.8.5.1 Modélisation structurelle

La modélisation structurelle consiste à décrire le composant ou le circuit par sa structure, c'est à dire par les éléments qui le décrivent (capacité, résistance, diode, ...etc.).

Le modèle structurel s'aligne ainsi sur la bibliothèque du fondeur ou du fabricant de circuit intégré. Il prend en compte les paramètres technologiques utilisés en fabrication. La modélisation structurelle utilise les sous circuits du simulateur et demande un temps d'analyse trop important lors de la simulation des systèmes complexes.

L'inconvénient vient de la taille des circuits. Certains circuits analogiques (amplificateur opérationnel) contiennent plusieurs centaines de transistors et autres composants ; ceci augmente considérablement le nombre de nœuds, et par la suite la taille de la matrice à traiter par le simulateur d'où le temps de calcul est grand [3].

I.8.5.2 Macromodélisation

La macromodélisation consiste à décrire le comportement d'un circuit par l'utilisation des primitives d'un simulateur. Le but essentiel de la macromodélisation est de réduire la taille du circuit et ainsi réduire le temps de simulation. En outre, il faut prendre garde à ce que le nombre de nœuds imposé par le macro-modèle ne dépasse pas celui qu'impose le circuit à modéliser, sinon il n'aura aucun profit sur l'encombrement et la taille des matrices qui sont générées, alors les modèles obtenus par cette approche permettent de réduire considérablement les temps de simulations [1].

L'objectif principal de la macromodélisation est de remplacer un système électronique ou une partie de ce système (une fonction ou un dispositif actif) par un modèle afin de réduire significativement le temps requis par les nombreuses simulations électriques effectuées en phase de conception. Pour ce faire, un macro-modèle doit répondre à deux exigences conflictuelles: il doit être structurellement le plus simple possible et en même temps simuler le comportement du circuit avec le maximum de précision.

Les macro-modèles sont construits à partir d'un nombre réduit de composants. Les composants utilisés sont des composants primitifs du simulateur. Nous pouvons inclure des éléments passifs (résistance, capacité, ...etc.), des sources dépendante et indépendante de type courant ou de tension linéaire ou non, statique, temporelles ou fréquentielles qui sont intégrés dans le module ABM (Analog Behavioral Modeling) de PSPICE [3].

La méthode de macromodélisation consiste à utiliser trois blocs de base composés par (Figure I.12) [5] :

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.

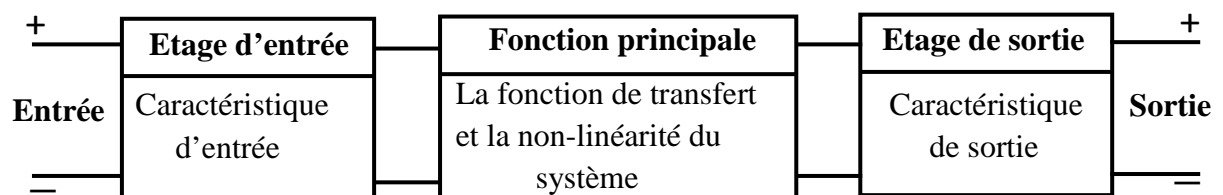


Figure I.12: Méthodologie de base de la macro-modélisation [5].

I.8.5.2.1 Avantage et inconvénient

L'avantage majeur de la macromodélisation, c'est qu'elle ne nécessite pas l'apprentissage d'un langage de programmation mais une bonne connaissance d'un simulateur analogique à base de SPICE, ces composants disponibles, et une bonne connaissance des fonctions à réaliser. Avec la macromodélisation, nous pouvons alors réaliser un grand nombre de fonctions : fonctions mathématiques, filtres, détecteurs de phase, modulateurs d'amplitude et de largeurs d'impulsions, oscillateur, convertisseur fréquence-tension, comparateurs, amplificateurs et boucle à verrouillage de phase.

Cependant, malgré les avantages et la simplicité apparente, la macromodélisation pose un certain nombre de limitations :

- La non-linéarité des composants et la tension de décalage dans la diode par exemple qui posent certains problèmes qui ne sont pas négligeables,
- Problème de convergence dû à la discontinuité ou au rebouclage des certains circuits,
- Paramétrage des composants qui n'existe pas dans SPICE de base, mais nous pouvons le trouver dans les dérivés de SPICE comme par exemple dans Eldo, SmartSpice™, PSPICE®, etc.
- Plage limitée du fonctionnement du macro-modèle, problème de paramétrage des composants (SPICE3),
- La limitation des circuits analogiques [5].

I.8.5.2.2 Méthodologie de la macromodélisation

Pour décrire de façon structurelle, SPICE a défini un langage pour la modélisation des fonctions analogiques appelé **macromodélisation**. 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 netlist de type SPICE est analysé par le simulateur lui-même pour construire un système d'équations basées sur les lois de Kirchhoff, donc, basées sur une loi de conservation d'énergie et sur les équations des composants. La macromodélisation 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 près du circuit initial. Dans un autre sens, la macromodélisation 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 macromodélisation 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 linéaire de bas niveau comme les diodes et les transistors bipolaires ou MOS de niveau 1 de SPICE.

Les sources contrôlées (sources dépendantes) sont des éléments idéaux qui permettent d'exprimer des relations mathématiques entre les courants et les tensions.

Il est possible de définir grâce à l'outil ABM de nombreuses équations non linéaires pour un courant ou une tension à partir de 2 primitives : EVALUE Source de tension contrôlée et GVALUE Source de courant contrôlée, sous la forme suivante [1]:

1/ *Ename* N+ N- *VALUE* = {*control expression*}

2/ *Gname* N+ N- *VALUE* = {*control expression*}

Une fonction "TABLE" permet également de définir les caractéristiques statiques de ces sources point par point :

1/ *Ename* N+ N- *TABLE* {*input expression*} = (*input1*, *Vout1*) (*input2*, *Vout2*) . .

2/ *Gname* N+ N- *TABLE* (*input expression*) = (*input1*, *Iout1*) (*input2*, *Iout2*) . . .

Alors SPICE définit quatre types de sources contrôlées :

- Source de tension contrôlée par une tension (VCVS), de la forme :

$$V_E = E V_C \text{ (linéaire),}$$

$$V_E = e(V_C) \text{ (non-linéaire).}$$

- Source de tension contrôlée par un courant (CCVS), de la forme :

$$V_H = H I_C \text{ (linéaire),}$$

$$V_H = h(I_C) \text{ (non-linéaire).}$$

- Source de courant contrôlée par un courant (VCCS), de la forme :

$$I_G = G I_C \text{ (linéaire),}$$

$$I_G = g(I_C) \text{ (non-linéaire).}$$

- Source de courant contrôlée par une tension (CCCS), de la forme :

$$I_G = G I_C \text{ (linéaire),}$$

$$I_G = g(I_C) \text{ (non-linéaire).}$$

Il est possible également de modéliser une fonction de transfert à partir de sa Transformée de

Laplace ou de son diagramme de Bode en amplitude et en phase. La première approche requiert l'utilisation de sources de type ELAPLACE et GLAPLACE :

1/ *Ename* N+ N- LAPLACE {input expression} = {S form expression}

2/ *Gname* N+ N- LAPLACE {input expression} = {S form expression}

Pour la deuxième approche, une fonction de type TABLE pour le domaine fréquentiel est aussi disponible : utilisation de sources de type EFREQ et GFREQ

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'.

La caractéristique non-linéaire des diodes est utilisée comme un opérateur conditionnel. Les diodes sont utilisées dans des comparateurs, des limiteurs de tension, etc.

Cependant, les diodes introduisent un décalage de tension (Figure I.13) et elle est de la forme :

$$V = N \cdot V_T \cdot \ln \left(\frac{I_D}{I_S} + 1 \right) \quad (\text{I.13})$$

Avec :

V : Tension au borne de la diode (Volt),

N : Coefficient d'émission,

V_T : Tension thermique ($k.T/q$) (Volt)

I_S : Courant de saturation (A)

I_D : Courant à travers la diode (A)

L'équation (I-13) montre que la tension aux bornes de la diode est affectée principalement par deux paramètres : N et I_S . Parmi ces deux paramètres, le coefficient d'émission (N) est le plus dominant. Ce coefficient représente un avantage pour réduire cette tension parasite. Quand le coefficient d'émission est inférieur à 1 la tension au borne de la diode est réduite (Figure I.13) [5].

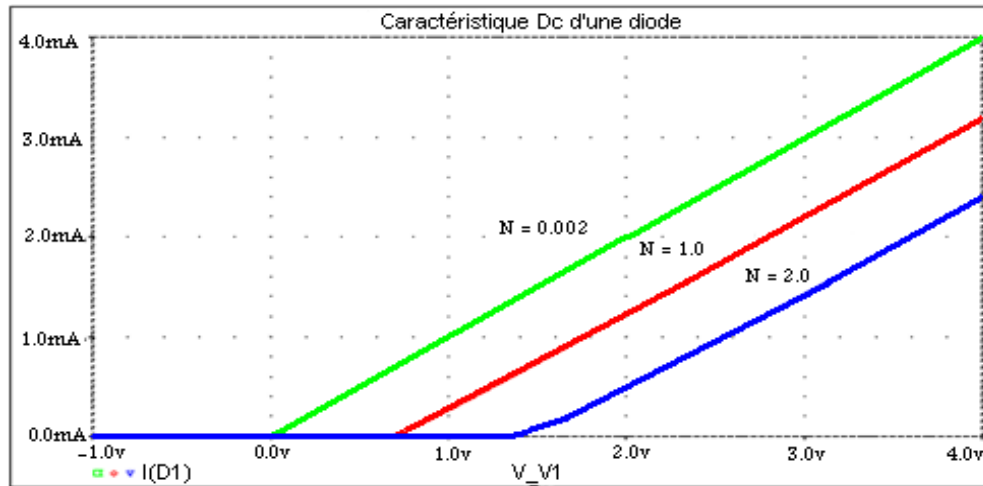


Figure I.13: Caractéristique d'un modèle SPICE d'une diode avec différentes valeurs de coefficient d'émission (N) [5].

Parfois, et en fonction des options de la simulation, cette méthode peut créer des problèmes de convergence. Pour remédier à ça, nous pouvons ajouter en série une source de tension fixe de même valeur que la tension de décalage.

I.8.5.3 Modélisation comportementale

La modélisation comportementale permet de réduire les temps de conception et de concevoir des circuits de plus grande qualité pour deux raisons essentielles :

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

I.9 conclusion

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

Les modèles seront d'abord réalisés dans le cas du composant idéal, puis nous introduisons progressivement les influences de chaque paramètre sur le comportement du composant ou du circuit.

Chapitre II

Langage de modélisation et simulateur

II.1 Introduction

La simulation électrique 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 électrique une nécessité absolue voir inéluctable.

L'objectif principal de la modélisation est d'expliquer le mode opératoire pour créer un modèle de composants à partir d'un schéma équivalent à base d'éléments *discrets* (*résistance, condensateur, bobine, diode, transistor...*) ainsi que de blocs fonctionnels (*Evalue, Gvalue, Etable, Sum, Diff, Integ ...*) puis définir un symbole équivalent associé à un macro modèle utilisable dans les simulations.

II.2 Choix du langage de modélisation

Après avoir déterminé, grâce aux études du précédent chapitre, les techniques et les niveaux de modélisation nécessaires, il nous a fallu déterminer le langage de modélisation le mieux adapté à nos besoins. Les langages susceptibles d'apporter une réponse aux problèmes de modélisation sont nombreux, mais peu répondent à nos attentes [6].

II.2.1 Langage de programmation orientée objet

Ces langages tels que C++, Visual Basic ou Java sont des langages évolués aux possibilités quasi-infinies du fait même qu'ils sont les briques de base pour la construction des applications. Les simulateurs en faisant partie, il est tout à fait possible d'utiliser ces langages à des fins de simulation. Cependant, puisque des simulateurs performants existent déjà, et étant donné qu'il est souvent possible de faire interagir les HDL avec le langage C++, il n'apparaît pas nécessaire d'en faire usage en tant que simulateur.

II.2.2 Langage de modélisation numérique

On retrouve dans cette catégorie les langages comme VHDL *{IEEE 1076-2000}*, Verilog *{IEEE 1364-2001}*, System C, Spec C et System Verilog. Ces langages très efficaces sont aujourd'hui associés à des outils qui permettent, moyennant une utilisation appropriée, la synthèse numérique de dispositifs complexes et le co-design pour les plus évolués.

II.2.3 Langage mathématique formel explicite

Un langage comme Matlab associé à Simulink permet de proposer aux utilisateurs une représentation graphique sous forme de boîtes noires pouvant être connectées les unes aux

autres et pouvant avoir des fonctions de transfert très complexes grâce aux possibilités mathématiques très poussées de Matlab. Dans cette conception de la modélisation, la gestion des équations implicites au sein d'un circuit (lois de Kirchhoff aux nœuds du circuit) n'est pas assurée. De ce fait, la conception d'un système optoélectronique complet à l'aide de Matlab/Simulink n'est pas envisageable simplement.

II.2.4 Langage de modélisation implicite dédiée à l'électronique

Dans cette famille incluant les langages comme SPICE, il n'est question que de modélisation électrique analogique. Il s'agit alors pour les concepteurs de modèles multi-domaines de transférer l'ensemble de leur dispositif dans le domaine électrique par analogie. Cette démarche est courante, mais peu pratique lorsque l'on sait que d'autres logiciels permettent de travailler directement dans les différents domaines de la physique. De plus, ce langage n'est pas capable d'assurer seul la modélisation des systèmes mixtes, ce qui n'est pas le cas de certains de ses concurrents.

II.2.5 Langage de modélisation analogique multi-domaines

Cette famille contient deux langages aujourd'hui supplantés par leurs "successeurs" : HDLA (qui s'est trouvé redirigé vers VHDL-AMS) et Verilog-A (qui a dû également s'orienter vers une technologie de simulation mixte sous la concurrence de VHDL-AMS avec Verilog-AMS).

Il existe cependant dans cette famille de langages le MAST. A l'origine propriétaire de Analog, passé depuis dans le domaine public, c'est un langage aux bases solides. En effet, depuis sa création en 1986, ce langage de simulation multi-domaines, mais uniquement analogique, a su conquérir les industries n'ayant pas recours au numérique, comme en mécanique ou en électronique de puissance.

Cependant, l'avènement des technologies numériques provoque de plus en plus d'incursions dans le domaine analogique (pour des structures de contrôle par exemple). De ce fait, les logiciels de simulation associés au langage MAST comme SABER incluent le support et l'interfaçage avec des langages comme VHDL.

Intéressant de devoir utiliser ce genre de langage propriétaire alors que certains sont conçus et normalisés pour les systèmes mixtes.

II.2.6 Langage de modélisation mixte multi-domaines

On retrouve ici les langages Verilog-AMS et VHDL-AMS. Ces langages dont les bases numériques se trouvent environ 20 ans en arrière, ont bénéficié récemment (1998 pour

Verilog-AMS et 1999 pour VHDL-AMS) d'extensions pour les signaux analogiques et mixtes (Analog and Mixed-Signal AMS). Comme leurs noms l'indiquent, ces langages permettent de traiter indifféremment des modélisations logiques, analogiques ou mixtes au sein d'un même composant ou système. Par ailleurs, la philosophie de conception de ces langages et leurs jeux d'instructions en font des langages intrinsèquement multi-domaines qui gèrent les équations implicites liées au fonctionnement d'un circuit.

II.3 Méthode de modélisation VHDL-AMS

De SPICE à VHDL-AMS ... et réciproquement

Au vu de l'étude précédente présentant l'état des outils associés, la conception mixte peut s'envisager, soit à partir d'un langage propriétaire, soit à partir d'un outil normé.

Même si VHDL-AMS est un langage de description comportementale de haut niveau, il existe plusieurs façons de décrire le fonctionnement d'un système ou circuit dans ce langage.

En particulier, un certain nombre de publications ont pour objet la traduction pure et simple d'un circuit SPICE structurel en VHDL-AMS. Par exemple, Kasula Srinivas a traduit des modèles SPICE purement électriques de composants actifs (diodes, transistors, bipolaires, transistors MOS) en VHDL-AMS sous forme d'équations différentielles. Les simulateurs actuels compatibles SPICE obtenant et résolvant directement ces dernières à partir de la netlist, le temps de simulation est identique pour les deux descriptions comme nous avons pu le vérifier sur quelques exemples. On peut alors s'interroger sur le bien fondé de cette simple traduction : elle ne profite par exemple pas des apports du langage en matière de description multi-technologique.

L'intérêt de VHDL-AMS dans le contexte précis de la modélisation d'un composant actif est justement de simplifier les modèles électriques en ne retenant que le strict nécessaire, et de les coupler entre autres à des modèles thermiques.

Réciproquement, alors que les simulateurs VHDL-AMS n'existaient pas encore, une idée mise en œuvre entre autres par Alali consistait à traduire purement et simplement les modèles VHDL-AMS en netlists SPICE, en matérialisant les équations différentielles par des composants passifs (résistances, inductances et capacités). Ici, l'aspect compatibilité ascendante avec VHDL'93 n'a pas été pris en compte.

II.4 OrCad PSPICE

II.4.1 Définition

OrCad PSpice est un logiciel de simulation mixte (analogique et/ou logique). La mise en œuvre d'une simulation repose principalement sur :

- une description des composants et des liaisons figurant sur un schéma, sous forme de fichier « Circuit »,
- Une description des signaux d'entrée appliqués sur le schéma sous forme de fichier « Stimulus »,
- une description des modèles de simulation des composants sous forme de fichiers « Modèles ».

Une bonne explication est donnée sur le schéma de la figure II.1

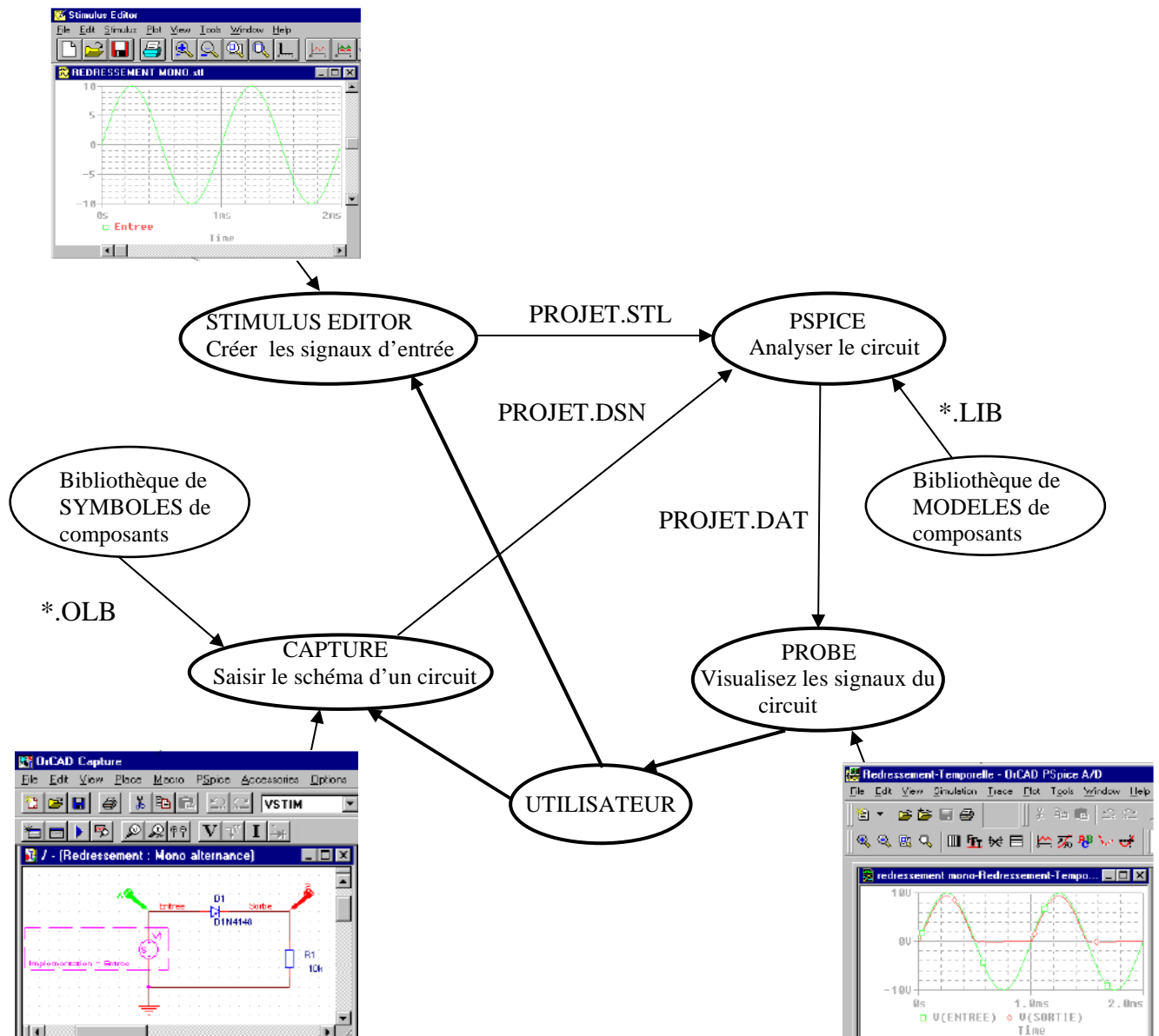


Figure II.1: description pour OrCad PSPICE [7].

PSPICE est un logiciel de simulation de fonctionnement de circuits électriques, initialement analogiques puis numériques ou mixtes. Il est issu du logiciel SPICE développé en FORTRAN en 1970 par D. OPEDEPERSON et L.W.NAGEL à l'université de Berkeley. Depuis cette époque ci le cœur du logiciel a peu varié il n'en est pas de même de son environnement. SPICE avait une interface de type texte et le résultat des simulations était sous forme de grands tableaux de chiffres imprimés sur de longs listings déroulés dans le crépitement des télétypes...

Pour simuler le fonctionnement d'un circuit électrique, analogique pour le moment, plusieurs étapes sont nécessaires :

- 1- Une description du circuit, les composants qui le constituent et leurs interconnexions. Ceci fait l'objet d'un fichier texte xxxx.CIR qui est soit écrit par l'utilisateur grâce à un quelconque éditeur de texte, soit construit automatiquement par le logiciel à partir d'un schéma tracé sur l'écran.
- 2- Une description des sources de tensions et courants qui sont reliées au circuit, alimentations et signaux d'excitation (STIMULI). Dans les versions DOS cette description est également contenue dans le fichier.CIR précédent.
- 3- Le calcul de la réponse du circuit par le logiciel SPICE proprement dit. Plusieurs types de simulation sont possibles nous les décrirons plus loin.
- 4- L'exploitation des résultats. Le temps des longs listings est révolu, les résultats sont présentés à l'écran sous forme de courbes multicolores [7].

II.4.2 Limitation de SPICE

SPICE (*Simulation Program with Integrated Circuit Emphasis*) a été développé à l'origine par l'Université de Californie, Berkeley. Le simulateur SPICE est considéré en réalité comme un standard de fait qu'on s'en sert pour analyser des circuits. Très vite il devient l'outil le plus efficace d'aide à la conception. Pourtant, il comporte des limitations dans certains domaines :

- **Modélisation mixte** : le simulateur SPICE est à temps continu, donc le modèle conçu doit être à temps continu. On a vu la nécessité d'un langage qui permette d'écrire des modèles mixtes pour tenir compte des temps continu et discret à la fois. SPICE ne peut pas supporter les représentations discrètes, et en conséquence, il n'est pas adapté pour la modélisation mixte, sauf au moyen d'une macro-modélisation lourde.

- **Modélisation comportementale** : la plupart du temps, c'est un avantage, d'une part, en terme de temps d'exécution et de mémoire demandée et d'autre part, pour simuler une partie d'un circuit dont le niveau de structure est très détaillé (structural) ou moins détaillé (comportemental). SPICE décrit explicitement le structural et décrit le comportemental implicitement pour un modèle analogique, en conséquence, le temps d'exécution est très long et il est très gourmand en mémoire pour le stockage des détails des composantes internes.
- **Transmission de données** : SPICE ne supporte que des systèmes conservatifs comme par exemple les circuits électriques qui obéissent aux lois de Kirchhoff (loi des nœuds et loi des mailles). En ce qui concerne les flots de données (non-conservatif), une façon de les représenter utilise le temps discret, or cette représentation n'est pas supportée par SPICE.
- **Transparence** : il est souvent nécessaire de connaître le détail primitif du modèle qui n'est pas explicité par le langage, pour pouvoir effectuer une représentation précise du système. Les modèles qui sont bâtis dans SPICE sont complexes et l'utilisateur ne peut pas contrôler les équations primitives qu'il contient, en conséquence, les modèles écrits en langage SPICE. En utilisant les modèles primitifs, c'est comme une boîte noire et il ne peut pas toujours décrire le comportement du système prévu [5].

II.5 VHDL-AMS

II.5.1 Histoire de VHDL-AMS

Né en 1987 du langage ADA (1979), VHDL *{IEEE 1076-1987}* est à l'origine un langage de modélisation de systèmes numériques. Complété et enrichi en 1993 *{IEEE 1076-1993 et IEEE 1164-1993}* et en 2000 *{IEEE 1076-2000}*, ce langage est basé sur une simulation événementielle, et non temporelle, des systèmes.

Avec les extensions *{IEEE 1076.3-1997, IEEE 1076.4-1995 et 1076.6-1999}* de VHDL et les outils qui y sont progressivement associés, les concepteurs de modèles arrivent jusqu'à la synthèse logique de circuits numériques à partir de simples morceaux de code, écrits de manière adéquate, décrivant les fonctions d'un système.

Alors que dans le même temps, d'autres langages, tels MAST(1986) et SPICE (1969), permettent de gérer des modélisations principalement analogiques, notamment dans les domaines de la mécanique et de l'électronique de puissance, il n'existe pas de langage permettant de modéliser les systèmes mixtes (numériques et analogiques) qui émergent avec l'apparition de l'électronique numérique de commande. Après une réflexion d'une dizaine

d'années associant chercheurs et industrielles, une extension dans le domaine analogique et mixte de VHDL est présentée en 1999 : VHDL-AMS [6].

II.5.2 Avantage 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.

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 multi domaines 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 grande force de ce langage est de permettre la simulation mixte en autorisant aussi bien les modélisations à temps continu (analogiques) qu'à événements 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.

Deux autres caractéristiques de VHDL-AMS, dont chacune est partagée avec quelques autres langages, sont le traitement des équations implicites (c'est-à-dire des équations où l'inconnue ne se trouve pas forcément dans le membre de gauche) et à travers celles-ci, l'utilisation des lois de Kirchhoff généralisées, fondement des relations implicites entre les différents nœuds d'un système.

II.5.3 Limite 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.

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. C'est le cas de l'instruction PROCEDURAL qui pourrait nous permettre de simuler un bloc d'instructions séquentielles à chaque ASP par exemple. La méthode de résolution matricielle des simulateurs empêche quant à elle d'implémenter une instruction comme DISCONNECT qui permettrait de retirer un élément du système, ce qui modifierait dans le même temps la structure de la matrice de calcul. Or cette opération ne peut être refaite en cours de simulation [6].

II.5.4 Evolution liée à VHDL-AMS

Avec la réaffirmation de la normalisation VHDL-AMS prévue en 2004, 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, présentée ci-après, 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. En effet, des logiciels comme Simplorer permettent de pallier à beaucoup de problèmes, comme l'absence du PROCEDURAL, par des moyens détournés comme l'utilisation de modèles C/C++ ou celle de composants utilisant le "méta-langage" SML de Simplorer qui peut communiquer avec le noyau de simulation VHDL-AMS et réagir sur ses ASP. Ceci a cependant l'inconvénient d'ôter la portabilité du système et il est peu probable qu'un palliatif puisse être trouvé aux problèmes du type DISCONNECT qui relèvent directement du fonctionnement du noyau de simulation. Les concepteurs font sans ces instructions pour l'instant, devront-ils continuer ainsi, avec à terme l'abandon de celles-ci dans la norme, ou bien les noyaux de simulation VHDL-AMS vont-ils évoluer ?

II.6 Choix du logiciel de simulation

Comme énoncé précédemment, notre choix s'est porté sur le 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.

II.6.1 Simulateur 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. 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.

Dans cette catégorie de simulateurs, ADVance MS (Mentor Graphics) est probablement le meilleur grâce à son implémentation la plus fournie de la norme VHDL-AMS et son moteur de simulation ELDO qui est un des plus rapide et dont les simulations convergent le mieux. Un avantage et une faiblesse d'ELDO est sa capacité à utiliser une bibliothèque modèle C propriétaire très performante qui permet de simuler très efficacement et très finement (avec les effets thermiques par exemple) des composants comme une diode ou un transistor. Ceci

nous a posé, et pose encore, de gros problèmes de portage pour les modèles développés avec ADVance MS vers Simplorer, et vice-versa.

Le logiciel de hAMSter, qui n'est plus distribué aujourd'hui (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.

II.6.2 Possibilité offerte par l'interfaçage avec VHDL-AMS

Important de pouvoir concevoir des modèles qui fassent cohabiter des composants de différents langages. Pour cela, il faut pouvoir les interfacer. La plupart des logiciels du marché permettent ces interactions, mais elles sont le fruit de démarches propriétaires. En effet, la norme VHDL-AMS ne donne aucune information sur la façon dont le langage devrait interagir avec d'autres. De ce fait, la portabilité des modèles complexes est extrêmement réduite.

Il serait intéressant que la norme définisse une façon d'interagir avec l'extérieur (un prototype d'appel de fonction ou une instruction propre par exemple), de manière à donner une direction aux développeurs de simulateurs. C'est cependant une tâche très ardue si l'on veut essayer de mettre au point un mécanisme simple qui permette d'accéder à n'importe quel langage étranger sans pour autant en limiter les possibilités [6].

II.6.3 Nouveaux simulateurs disposant d'un (*Graphical User Interface*)

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.

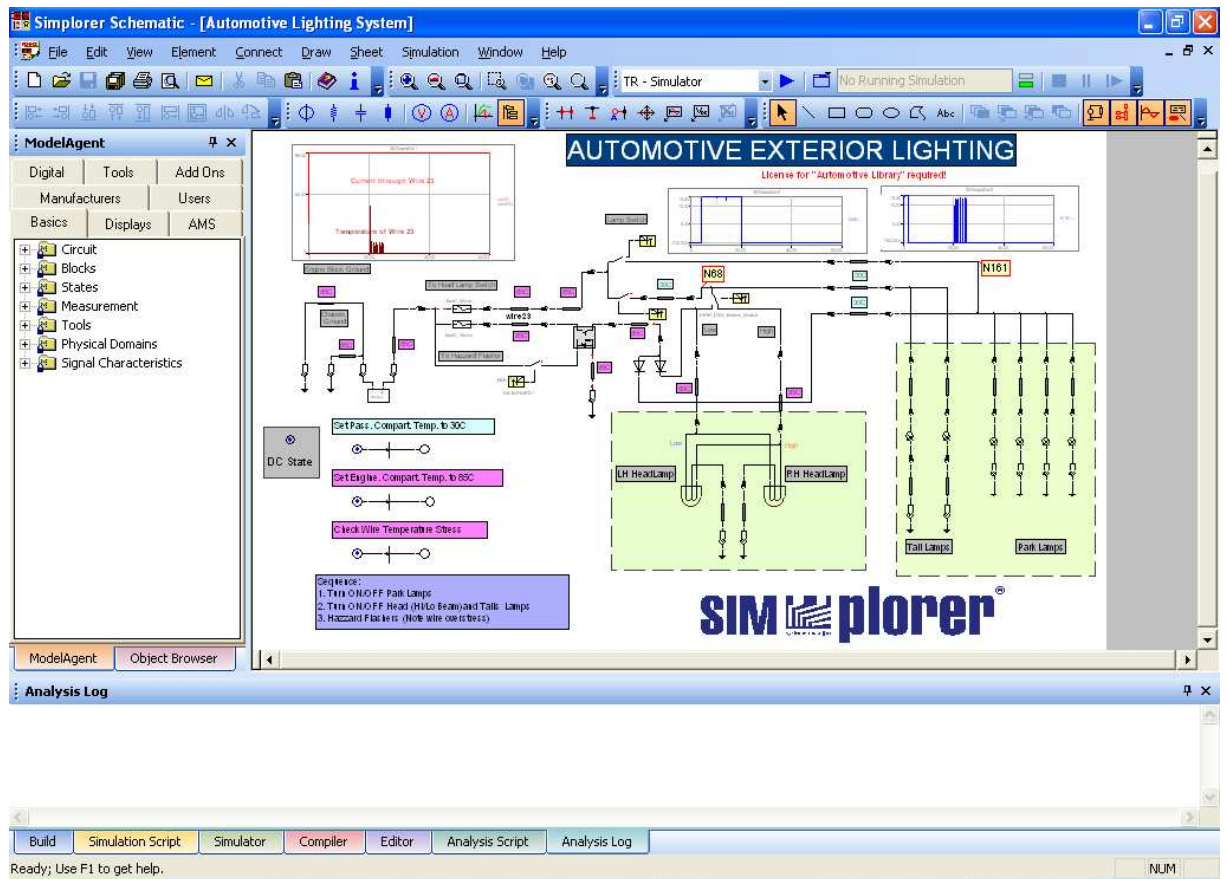


Figure II.2: Vue schématique d'un circuit multi-abstractions, multi-domaines dans Simplorer V7.

Il existe actuellement trois produits suffisamment puissants pour prétendre à une utilisation industrielle :

- ADVance AMS de la société Mentor Graphics
- hAMSter (*H*igh performance *AMS* tool for *E*ngineering and *R*esearch)
- SMASH de la société Dolphin

II.7 Simplorer

II.7.1 Introduction

Ce simulateur appartenant à la dernière génération utilise un noyau de simulation VHDL-AMS issu du simulateur hAMSter, 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 primitives 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 manœuvre 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.

II.7.2 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 (*figure II.3*) que l'interface des feuilles de schémas permettant de connecter et de faire interagir les différents éléments d'un système :

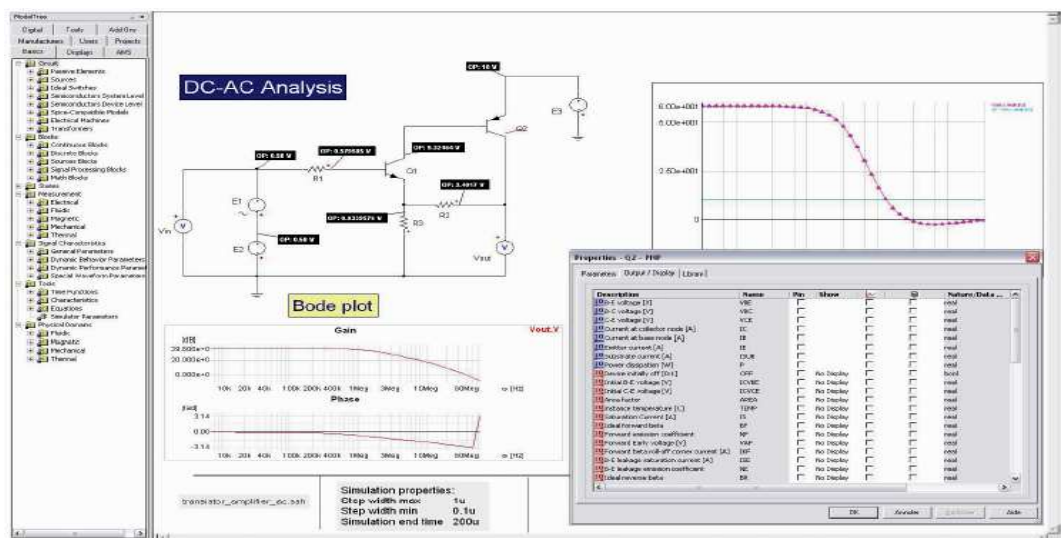


Figure II.3: Quelques aspects de l'interface graphique utilisateur [6].

Les gestionnaires de modèles et de symboles permettent en cela de leur associer des représentations graphiques, statiques, symboliques et explicites.

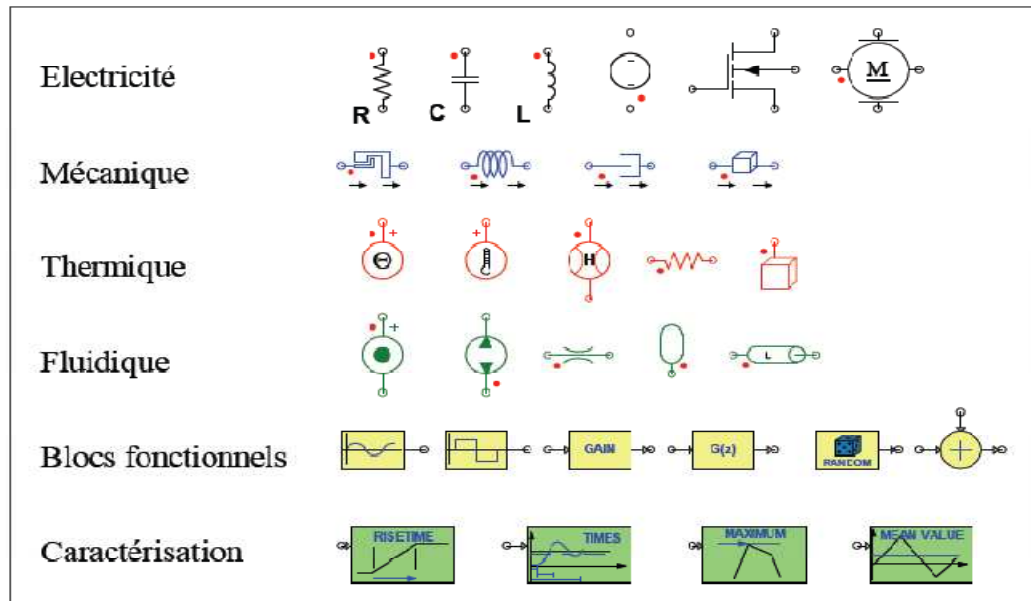


Figure II.4: Exemple des symboles extraits de la bibliothèque standard de Simplorer et natures ou domaines associés [6].

Dans le cadre d'un langage comme VHDL-AMS qui permet de simuler différents champs de la physique en interdisant les connexions abusives entre domaines, il peut être intéressant d'avoir recours à un code de couleurs dépendant de la "NATURE" des ports utilisée. Cette possibilité est offerte par défaut dans Simplorer (**figure II.4**), tout en offrant à l'utilisateur la faculté de personnaliser complètement ce jeu de couleurs, mais également celles de tout élément du schéma. Cette option est très intéressante lorsque l'on veut faire ressortir des portions de circuit qui ne peuvent être traitées en utilisant les terminaux [6].

II.8 Modèle VHDL-AMS

II.8.1 Structure d'un modèle VHDL-AMS

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**.

L'**ENTITY** est la partie (ou l'interface) qui communique entre le monde extérieur et le modèle au moyen de deux objets : **GENERIC** et **PORT**. Nous pouvons comparer l'**ENTITY** à une boîte noire où seuls les nœuds sont visibles, une partie de ces nœuds sont les ports d'entrée/sortie. Les **GENERICs** sont des constantes (ou des variables statiques), les paramètres, qui peuvent être modifiés par la suite. Les **PORTs** sont les variables ou les nœuds dynamiques. 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.

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 *ARCHITECTURE*s qui lui font appel avec différents types de description et pour une *ARCHITECTURE* donnée, il y a une et une seule *ENTITY*. (**Figure II.5**) [5].

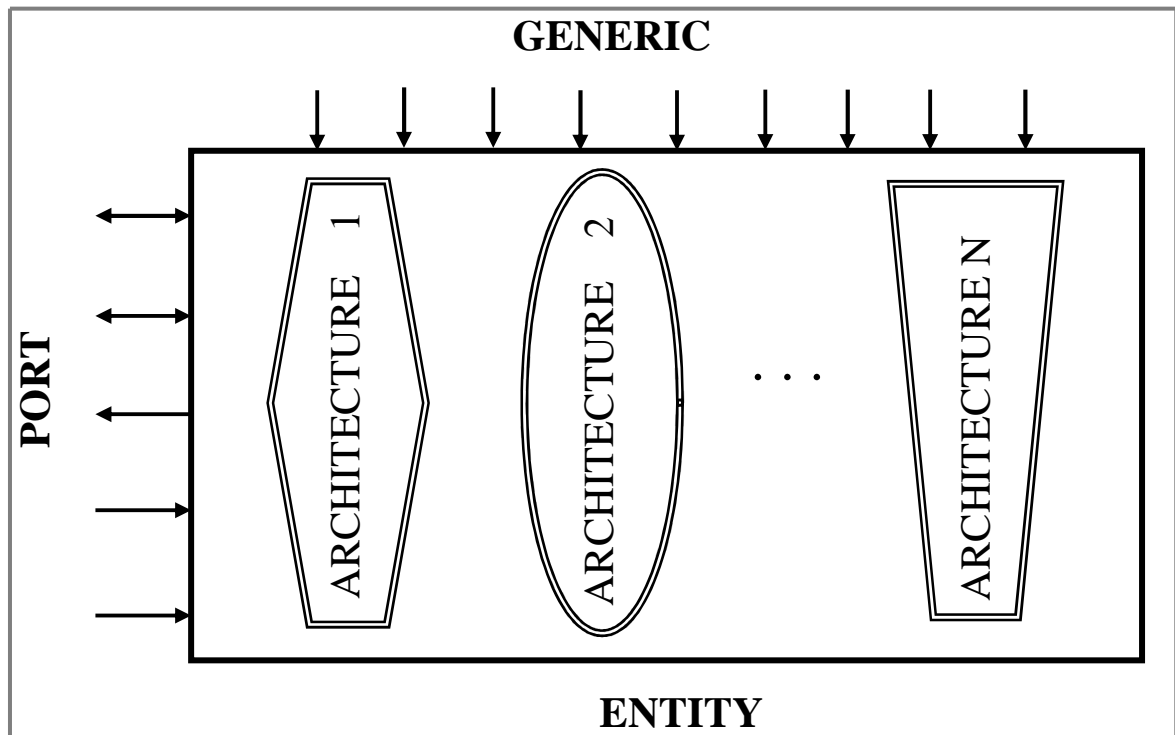


Figure II.5: Structure d'un modèle VHDL-AMS [5].

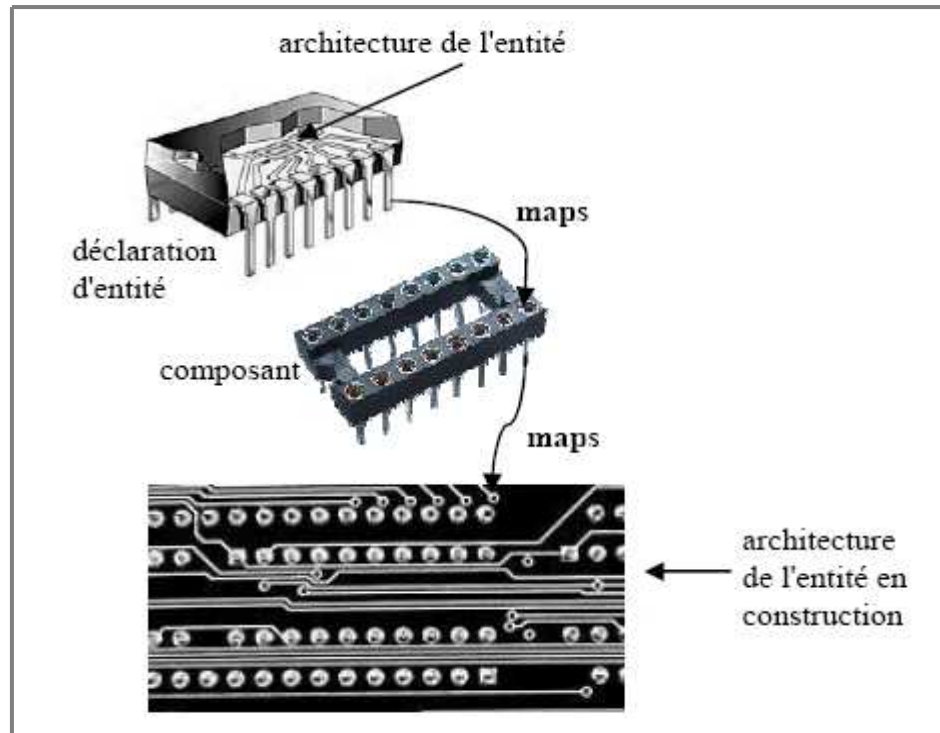


Figure II.6: Instance du composant et sa configuration avec une entité demandant deux « port map » et deux « generic map » [8].

Syntaxe :

```

ENTITY <nom_entite> IS
    GENERIC ( <declaration_GENERIC_1>;
              <declaration_GENERIC_2>;...
              <declaration_GENERIC_N>;
    PORT (<declaration_PORT_1>;
           <declaration_PORT_2>;...
           <declaration_PORT_N>;);
    [<declarations_variables_globales>;]
[BEGIN
    <controle_parametres_entree>]
END [ENTITY]<nom_entite>;

ARCHITECTURE <nom_arch_1> OF <nom_entite> IS
    <declarations_fonction_procedure>;
    <declarations_constantes>;
    <declarations_terminaux>;
    <declarations_types>;
    <declarations_variables>;
BEGIN
    <type_modele>;
END [ARCHITECTURE] <nom_arch_1>;
ARCHITECTURE <nom_arch_2> OF <nom_entite> IS...
ARCHITECTURE <nom_arch_N> OF <nom_entite> IS...

```

II.8.2 Classe d'objet

II.8.2.1 Quantité

Les inconnues des DAE introduites précédemment sont appelées les **quantités** (*QUANTITY*) et ces dernières constituent de nouvelles classes d'objets dans VHDL-AMS, ces objets retiennent l'information jusqu'à la prochaine période de résolution. Mais, les quantités prennent leurs valeurs comme résultat d'une résolution qui est effectuée dans le solveur analogique. Ce mécanisme est propre aux quantités et de cette façon, ils constituent une classe distincte d'objets. Les quantités doivent être de type réel flottant.

Syntaxe :

QUANTITY <quantity_1>,...,<quantity_N> : <type ou sous-type>;

Beaucoup d'opérations sont associées aux quantités, par exemple les dérivées et les intégrales qui sont définies dans VHDL-AMS comme des attributs prédéfinis. Pour chaque quantité Q : on note $Q'\text{dot}$ pour désigner la quantité du même type que Q qui représente la dérivée de Q en fonction du temps. On note aussi $Q'\text{Integ}$ pour désigner une quantité du même type que Q qui représente l'intégration de Q en fonction du temps de 0.0 à l'instant courant. Pour les dérivées des ordres supérieurs, il suffit d'écrire : $Q'\text{dot}'\text{dot}$ (ordre 2). Il en est de même pour les intégrales avec $Q'\text{Integ}'\text{Integ}$ (ordre 2) [5].

II.8.2.2 Terminal et Nature

Les **TERMINALS** fournissent des points de connexion interne au modèle : au niveau architecture ou externe : au niveau entité (*PORT*). Les terminaux sont souvent utilisés pour les systèmes conservatifs car les branches sont utilisées pour des systèmes interconnectés donc, des systèmes qui suivent une loi de conservation d'énergie. Rappelons que pour les systèmes non conservatifs, la connexion se fait à travers les quantités (les inconnues), de ce fait, il apparaît un système couplé par des quantités d'où la création des équations différentielle et algébrique pour décrire le comportement d'un système dynamique. Pour les systèmes conservatifs, les quantités ne sont pas associées directement mais à travers les terminaux. Les terminaux ne contiennent aucune valeurs mais ce sont les quantités associées qui l'ont pour former le DAE, donc, son rôle est de faciliter la description des systèmes conservatifs.

Syntaxe :

TERMINAL <Node1>,...,<NodeN> : <struc_nature>; -- nœud interne (architecture)

PORT (TERMINAL <Node1>,...,<NodeN> : <struc_nature>); -- nœud externe (entity)

Les quantités associées sont appelées *quantité de branche* (*branch quantity*) qui se décomposent en deux éléments la quantité aux bornes de la branche (*across quantity*) qui représente l'effort lié à l'effet et la quantité à travers la branche (*through quantity*) qui représente le flot lié à l'effet. La quantité aux bornes ou à travers une branche forme une structure de type *nature* (domaine physique ou discipline). Les natures se décomposent en deux : *nature simple* ou *nature composé* avec un sous-élément scalaire qui est de nature simple : Tableau II.1

Nature	Effet	Effort
Electrique	Tension	Courant
Thermique	Température	Puissance
Hydraulique	Pression	Débit
Mécanique de translation	Vélocité	Force
Mécanique de rotation	Angle de rotation	Torsion

Tableau II.1: Effort et effet pour différents domaines [5].

Syntaxe :

```

QUANTITY <Effort_Quantity> ACROSS
<Flot_Quantity> THROUGH <Node1> TO <Node2>;

    SUBTYPE <Subtype_Effort> IS REAL;
    SUBTYPE <Subtype_Effet> IS REAL;

NATURE <struc_nature> IS
    <Subtype_Effort>across
    <Subtype_Effet>through
    <Nature_Ref>reference;
```

La référence doit être fournie parce que toutes les quantités *across* sont différentielles et elles doivent avoir une référence stable qui est considéré comme le zéro, donc on doit choisir pour chaque domaine une référence qui soit différente de toutes les autres références.

II.8.3 Déclaration simultanée

VHDL-AMS ajoute aux deux types d'instruction de VHDL, séquentielle et parallèle, un nouveau type d'instruction pour les équations différentielles et algébriques :

L'*instruction simultanée*. Elle inclut les expressions VHDL ordinaire qui peuvent être évaluées de manière ordinaire mais l'interprétation sur la valeur résultante et l'effet sur le comportement des objets supportant ces valeurs est nouveau.

Les instructions simultanées peuvent avoir lieu partout où un signal parallèle peut être assigné. La forme de base de la déclaration est la suivante :

Syntaxe :

```
[label :] simple_expression == simple_expression_1;
```

Par exemple l'équation décrivant une résistance pourrait être écrite $I = V/R$; où I est une quantité représentant le courant qui traverse, V une quantité représentant la tension aux bornes, et R est la valeur de la résistance. Les quantités peuvent avoir des composantes : dans ce cas chaque sous-élément de gauche doit avoir un sous-élément correspondant à droite. Les expressions peuvent se référer à une quantité, un signal, une constante, variable partagée, fonction. Lorsque le calculateur (solveur) analogique a correctement établi les valeurs de chaque quantité, les sous-éléments correspondants seront approximativement égaux.

Le langage définit deux formes d'instructions simultanées :

- La déclaration simultanée procédurale qui fournit à l'utilisateur la notation DAE avec le style séquentiel. Il peut inclure toutes les déclarations du langage VHDL sauf les déclarations d'attentes (*wait*) et de signaux (*signal*),

Syntaxe :

```
[procedural_label:]  
PROCEDURAL [IS]  
    <partie_declaration_procedural>;  
BEGIN  
    <Instructions_procedural_simultanee>;  
END PROCEDURAL [procedural_label];
```

- Les déclarations simultanées *if* et *case* supportent la description comportementale. Chacune contient une liste arbitraire de déclarations simultanées dans la partie déclarative.

Syntaxe :

```
[if_label :]  
IF <condition_1>USE  
    <partie_instruction_simultanee_1>;  
  
[ELSIF <condition_2>USE  
    <partie_instruction_simultanee_2>;]  
  
ELSE  
    <partie_instruction_simultanee_3>;  
END USE [if_label];
```

```

[case_label :]
CASE <expression> USE
    WHEN => choix_1
        <partie_instruction_simultanee_1>;

    WHEN =>choix_N
        <partie_instruction_simultanee_N>;
END CASE [case_label];

```

Chaque expression caractéristique correspond à une expression de la forme $F(x, dx/dt, t)$.

Chaque déclaration simple est une collection d'expressions caractéristiques, une pour chaque sous-élément scalaire de l'expression.

Le calculateur analogique détermine la valeur de chaque quantité de manière à ce que les valeurs des expressions caractéristiques soient proches de 0 et résolvent alors les DAEs du modèle. Les algorithmes numériques utilisés par le solveur analogique produisent une solution exacte. Chaque expression caractéristique comme chaque quantité appartient à un groupe de tolérance (**TOLERANCE1**) qui est hérités de son sous-type. Donc là, les tolérances sont utilisées pour déterminer la précision exigée des solutions délivrées par le solveur. Si une tolérance est déclarée en même temps qu'on déclare la quantité ou dans la déclaration des sous-types, sinon, on peut le mettre à droite d'une expression simultanée et le résultat dans les deux cas, est la tolérance d'une des quantités ou celle de l'ensemble [5].

Syntaxe :

```
SUBTYPE <variable_1> IS REAL TOLERANCE "default_variable_1";
```

```
QUANTITY <variable_q_1> : variable_1;
```

Ou :

```
QUANTITY <variable_q_N> : REAL TOLERANCE "default_variable_q";
```

Ou :

```
<simple_expression> == <simple_expression_1> TOLERANCE "default_variable_q";
```

Si une discontinuité apparaît dans la solution du DAE, la structure mathématique l'indique au calculateur. Ce mécanisme d'arrêt (**break**) sert ce but : un modèle doit générer un pseudo-événement à chaque temps où intervient une discontinuité. Une discontinuité apparaît si une quantité est assimilée à un signal dans une simple déclaration et qu'un événement intervient sur ce signal. Il n'y a aucun algorithme connu qui peut de manière sûre et efficace détecter et corriger correctement les discontinuités sans connaissance du moment d'apparition. Par conséquent, le langage inclut une déclaration d'arrêt (**break**) qui a une forme

séquentielle et simultanée. L'exécution d'un *break* crée un événement sur le signal *break* implicite, ce signal n'est pas visible dans le modèle. La sémantique du langage requiert que le calculateur assume la discontinuité quand le processus assigné au signal *break* est actif. La déclaration de *break* inclut une possibilité pour la spécification des nouvelles conditions initialisées pour les quantités spécifiées, qui seront appliquées juste après la discontinuité. On peut mettre aussi les conditions initiales au même instant que la déclaration du *break*.

Syntaxe :

```
[label :]  
BREAK [FOR quantity_1] USE quantity_2 => expression1]  
[ON signal_1] [WHEN condition];
```

L'exemple suivant démontre l'utilisation d'un arrêt dans un convertisseur simple N/A :

```
ENTITY dac IS  
    GENERIC ( Vhigh : REAL := 0.5 );  
    PORT (SIGNAL S : IN Bit;  
          TERMINAL A : OUT Electrique );  
END ENTITY dac;  
  
ARCHITECTURE arch_dac OF dac IS  
    QUANTITY V across I through A;  
BEGIN  
    IF S = '0' USE  
        V == 0.0;  
    ELSE  
        V == Vhigh;  
    END USE ;  
BREAK ON S;  
END ARCHITECTURE arch_dac;
```

II.9 Apport de VHDL-AMS

Les caractéristiques de ce langage, par rapport à d'autres sont décrites ci-après :

▪ La modélisation mixte :

- *Quantity* et *Terminal* sont deux objets introduits pour décrire la simulation comportementale à temps continu. *Quantity* décrit le résultat d'une fonction analytique en fonction du temps. *Quantity* peut être utilisé en conjonction avec l'objet à événement discret *signal* pour représenter la modélisation mixte d'un système. *Terminal* représente les nœuds des branches *across* ou *through*.

- L'instruction *Break* dans VHDL-AMS est utilisée pour exprimer la discontinuité dans une simulation à temps continu et couramment utilisée comme moyen de communication (ou synchronisation) entre la simulation à temps continu et discret.
- **La modélisation comportementale :**
 - Le comportement d'un système continu peut être décrit par un ensemble d'équations algébriques et différentielles (DAE : *Differential and Algebraic Equations*) et par des instructions simultanées. Ces instructions simultanées expriment formellement les DAEs qui déterminent les valeurs des quantités du modèle.
 - *Signal* (quantité numérique) est utilisé dans la description d'un système à temps discret.
 - Les quantités implicites, comme \dot{Q} et Q_{integ} expriment le comportement dynamique (respectivement dérivation et intégration) de quantité qui leur sont associés.
- **Transmission de données :** VHDL-AMS supporte les systèmes conservatifs (loi de *Kirchhoff* pour les circuits électriques) pour modéliser les systèmes physiques qui sont représentés par les quantités et non-conservatifs pour modéliser le flot de données d'un système qui est représenté par les signaux (*signal*). Les deux types forment le système mixte.
- **Modélisation mixte et multi-technologie :** VHDL-AMS ne supporte pas seulement le domaine électrique, mais aussi n'importe quel système physique (hydraulique, thermique etc.) qui peut être décrit en utilisant les équations algébriques et différentielles ou en utilisant d'autres langages qui peuvent s'interfacer avec VHDL-AMS (ex : C, System C, Verilog- (AMS), ModelSim etc.). La résolution de ces systèmes doit inclure la gestion des discontinuités. D'autre part, il faut respecter les exigences au niveau des interactions entre partie numérique et partie continue des systèmes mixtes. *Nature* représente le domaine technologique pour les systèmes conservatifs. *Across*, *through* et *quantity* préservent la loi de conservation dans le système physique.
- **La transparence :** VHDL-AMS n'a pas de modèles primitifs (prédéfinis), qui sont déjà implantés. Le concepteur possède la flexibilité de modéliser ses propres systèmes comportementaux ou structurels, et l'utilisateur possède la liberté de modifier les modèles pour les adapter à ses besoins [5].

II.10 Application à la Modélisation hiérarchique

La conception de systèmes selon les méthodes descendantes (*top-down*) et montantes (*bottom-up*) peut être menée avantageusement avec VHDL-AMS en considérant sa capacité de modéliser à différents degrés d'abstraction comme indiqué sur la (figure II.7).

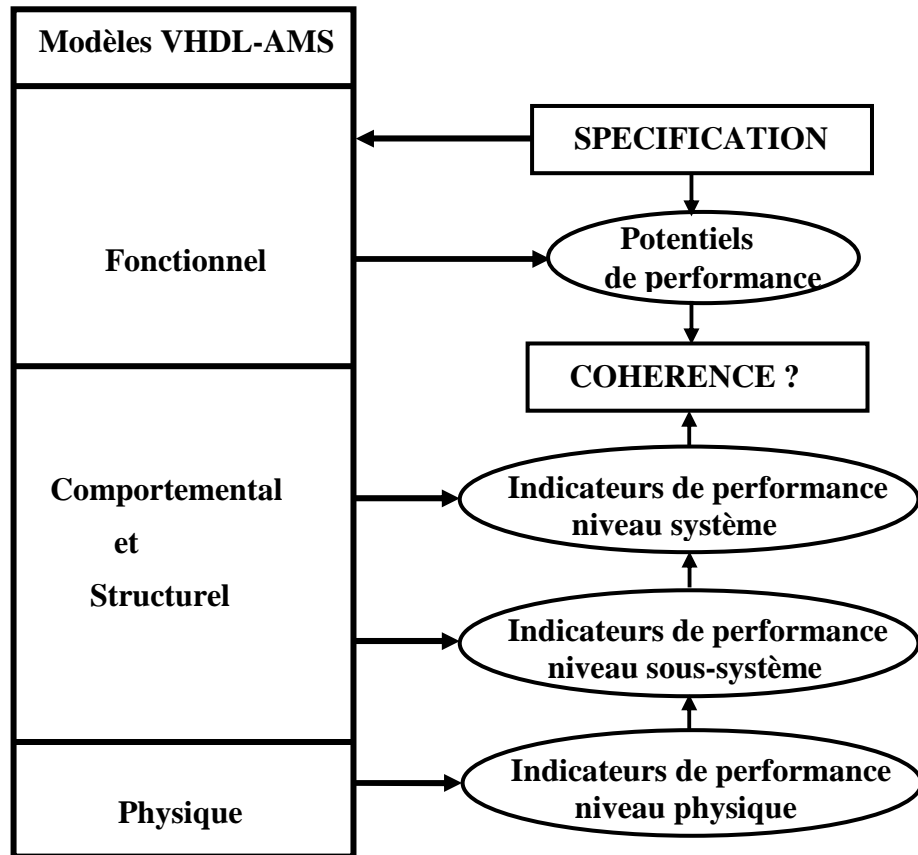


Figure II.7: Méthodologie de conception *top-down* et *bottom-up* avec VHDL-AMS [5].

La méthodologie de conception schématique hiérarchique proposée permet :

- de pouvoir « descendre », en suivant le processus descendant (*top-down*), dans les différents niveaux d'abstraction nécessités par le projet de conception.
- de remonter, en suivant le processus montant (*bottom-up*).

Ainsi, la méthodologie de conception de systèmes multi-technologique proposée dans nos précédentes publications. Elle se résume ainsi :

On recherche une optimisation entre les approches montantes et descendantes pour fournir un modèle cohérent entre son niveau physique et son niveau fonctionnel.

Pour cela :

- on divise le système en différents sous-systèmes comportementaux,

- on recherche des indicateurs de performance pour chaque niveau de modélisation,
- on répercute progressivement l'impact des indicateurs de performances vers le système complet [5].

II.11 Conclusion

Dans ce chapitre, nous avons décrit le cheminement qui nous a conduits à élaborer un outil spécifique à base de VHDL-AMS et de SPICE. Nous avons vu aussi les inconvénients du SPICE et les avantages de VHDL-AMS. Nous nous apercevons que le simulateur SPICE ne peut pas entièrement répondre à ces questions, que le langage VHDL-AMS.

Ces simulateurs doivent faire face à des problèmes de convergence qui apparaissent essentiellement lors de la recherche du point de fonctionnement pour les circuits comportant un nombre élevé de transistors, ainsi que pour les circuits fortement couplés. Alors les capacités de convergence des simulateurs analogiques sont fortement conditionnées par le nombre de transistors. Même si de nouvelles techniques de simulation algorithmes de résolution 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.

Etant donné la position que semble prendre VHDL-AMS au sein des entreprises, l'évolution des simulateurs dans le sens de la dernière génération devrait se voir renforcée avec une prise en charge plus complète et plus centrale de VHDL-AMS. Cette tendance s'observe déjà au sein de Simplorer 7.0 qui propose une meilleure implémentation de la norme et qui a uniformisé ses interfaces de composants quel que soit le langage utilisé.

Chapitre III

Simulation des circuits électriques



III.1 Introduction

Ce chapitre montre comment un langage de description structurelle, tel que le langage d'entrée du simulateur SPICE, nommé lui-même SPICE par simplification et le langage VHDL-AMS, peut être utilisé pour la modélisation comportementale des primitives du simulateur qui sont les éléments actifs comme les diodes et les transistors (bipolaire, MOS, JFET) et les éléments passifs comme les résistances, les capacités, les inductances ce sont des éléments idéaux qui permettent d'exprimer facilement des relations mathématiques entre tensions et courants dans le domaine fréquentiel ou temporel.

III.2 Connexion analogique

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

- ✚ Les *modèles de bas niveau* pour lesquels les “impédances” d'entrée/sortie doivent être prises en compte, sont caractérisés par des interconnexions “physiques” qui obéissent aux lois de Kirchhoff généralisées,
- ✚ Les *modèles de haut-niveau*, utilisés dans des *schémas-blocs*, sont liés entre eux par des grandeurs mathématiques qui n'appartiennent pas nécessairement à un domaine physique particulier [4].

III.3 Connexion physique

On définit de manière générale deux types de variables pour chaque domaine physique:

- ✚ Le type *potentiel (across)* : la valeur des variables de ce type est propre à chaque nœud du réseau et est définie par rapport à un nœud de référence (un pour chaque domaine).
- ✚ Le type *flux (through)*: il désigne les grandeurs relatives à une branche comprise entre deux nœuds. Un flux est d'autre part une grandeur absolue et unidirectionnelle.

Nous présentons dans ce qui suit les quelques composants modéliser par PSPICE et leur modèle en VHDL-AMS tels que : la résistance, capacité et l'inductance.

Les capacités et inductances sont utilisées pour réaliser des opérateurs de dérivation et d'intégration qui sont ensuite utilisés pour décrire des fonctions de transfert en s [4].

La fonction de transfert du filtre par exemple
$$H(s) = \frac{V_{out}}{V_{in}} = \frac{G.R}{1+R.C.s} \quad (III.1)$$

III.3.1 Resistance

La modélisation d'une résistance par PSPICE est effectuée avec le bloc ABM Gvalue comme indiqué dans la figure (III.1).

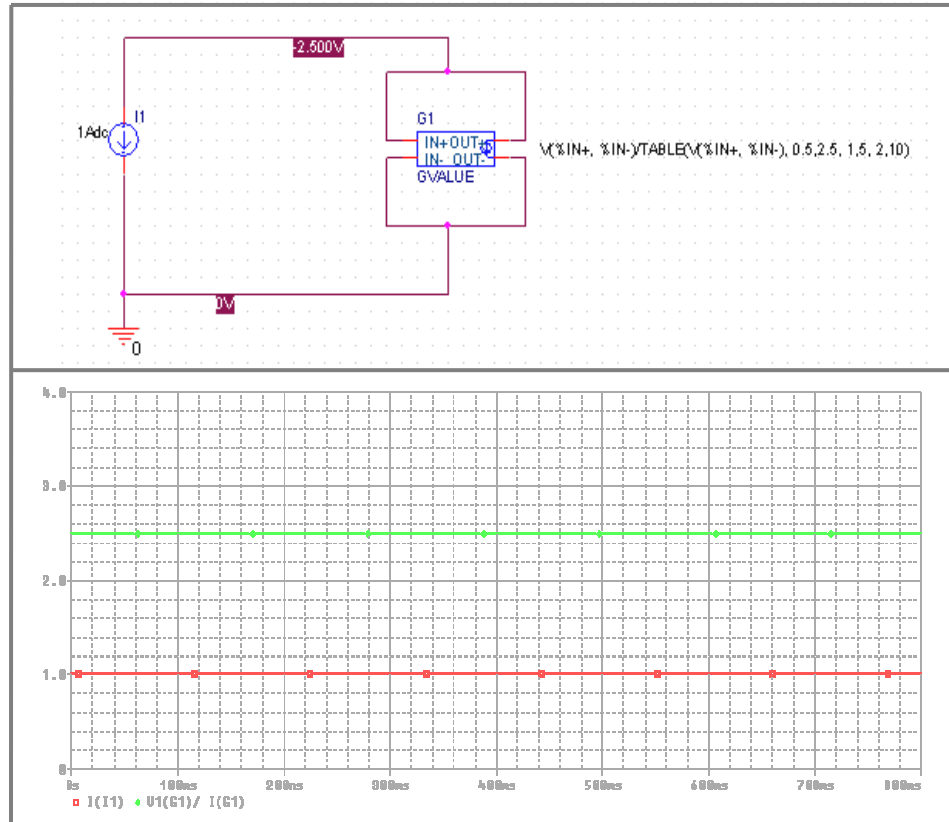


Figure III.1: Modélisation et simulation par PSICE d'une résistance.

L'équation suivante représente la relation entre la tension et le courant qui est la loi d'ohm :

$$V(t) = R \cdot i(t) \quad (\text{III.2})$$

La figure suivante décrit le code VHDL-AMS et la représentation graphique d'une résistance par Simplorer V7.0.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY R IS
    GENERIC (R : REAL := 1.0e+3);
    PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY R;

ARCHITECTURE behav OF R IS
    QUANTITY V ACROSS I THROUGH p TO m;
BEGIN
    V == I*R;
END ARCHITECTURE behav;

```

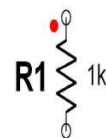


Figure III.2: Modèle VHDL-AMS et représentation graphique associée d'une résistance (Simplorer 7.0).

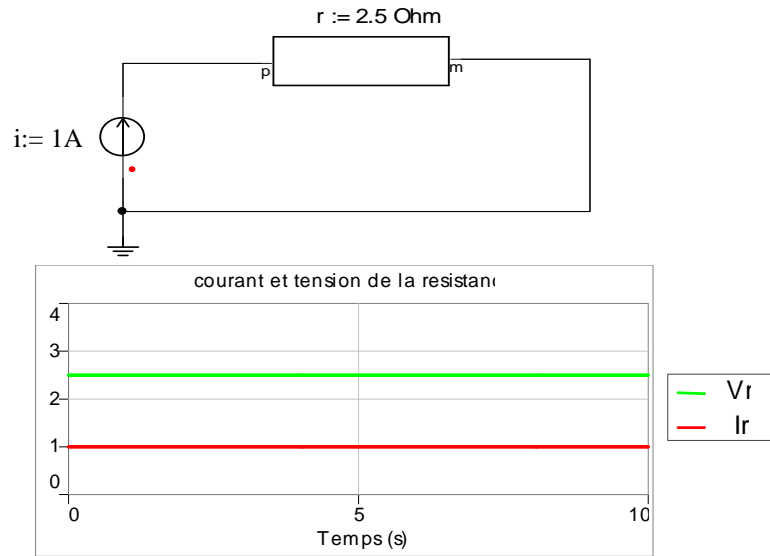


Figure III.3: Modélisation et simulation par Simplorer d'une résistance.

On conclut que les résultats de simulation sont identiques pour les deux applications.

Une autre méthode permet de grouper tous les composants dans un seul programme et les résultats de simulation sera le même.

III.3.2 Capacité

L'équation suivante représente la relation entre la tension et le courant d'une capacité :

$$i(t) = C \frac{dV(t)}{dt} \quad (\text{III.3})$$

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY C IS
    GENERIC (V0: VOLTAGE := 0.0;
             C : CAPACITANCE := 1.0e-6);
    PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY C;

ARCHITECTURE behav OF C IS
    QUANTITY v ACROSS i THROUGH p TO m;
    QUANTITY temp_charge : CHARGE := C*V0;
BEGIN
    IF (domain = quiescent_domain) USE
        temp_charge == C*V0;
        V == V0;
    ELSE
        temp_charge == C * v ;
        I == temp_charge'dot;
    END USE;
END ARCHITECTURE behav;

```

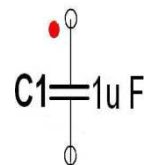


Figure III.4: Modèle VHDL-AMS et représentation graphique associée d'une capacité (Simplorer 7.0).

L'instruction ('dot) présente l'opération de dérivation en VHDL-AMS.

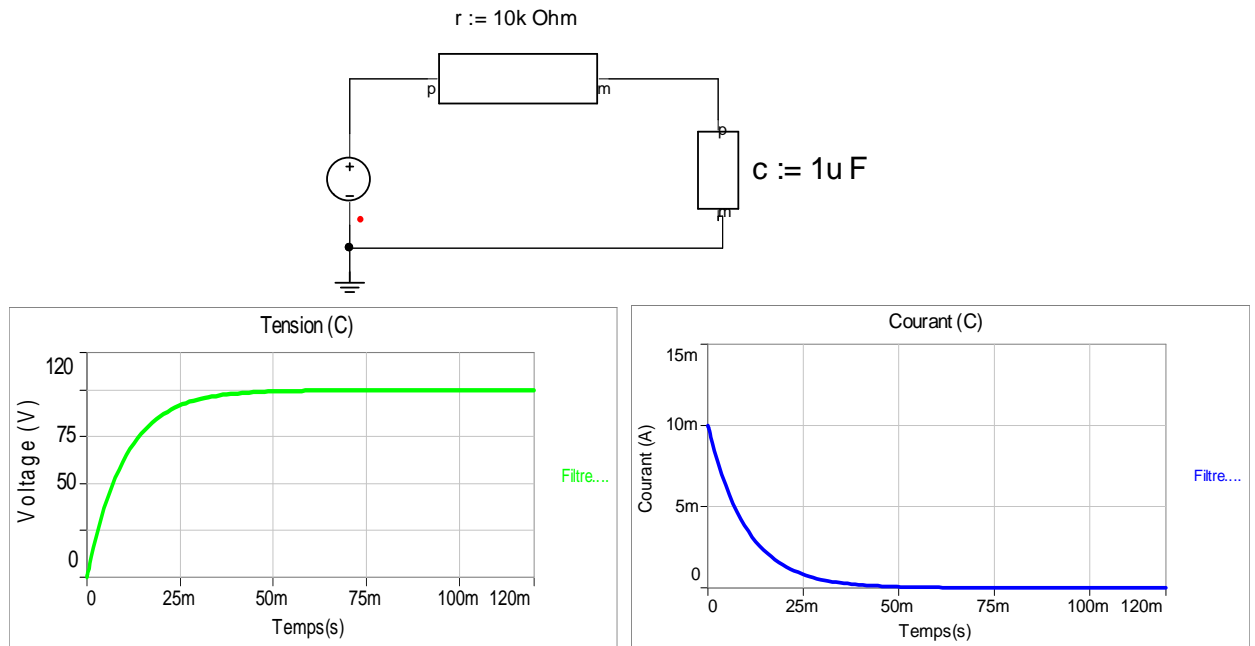


Figure III.5: Modélisation et simulation par Simplorer d'un filtre.

Les deux boîtes de r et c de la figure III.5 contient un code en VHDL-AMS c'est-à-dire le premier boîtier contient le code VHDL-AMS d'une résistance et la deuxième contient le code d'une capacité en VHDL-AMS.

III.3.3 Inductance

La relation entre la tension et le courant d'une inductance est donnée par :

$$V(t) = L (i(t)) \frac{dV(t)}{dt} \quad (\text{III.4})$$

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY L IS
  GENERIC (I0 : CURRENT := 0.0;
           L : INDUCTANCE := 1.0e-3);
  PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY L;

ARCHITECTURE behav OF L IS
  QUANTITY V ACROSS I THROUGH p TO m;
  QUANTITY flx : FLUX := L*I0;
BEGIN
  IF (domain = quiescent_domain) USE
    flx == L * I0;
    i == I0;
  ELSE
    flx == L * i;
    V == flx'dot;
  END USE;
END ARCHITECTURE behav;

```

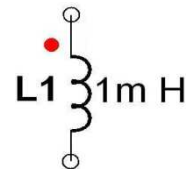


Figure III.6: Modèle VHDL-AMS et représentation graphique associée d'une capacité (Simplorer 7.0).

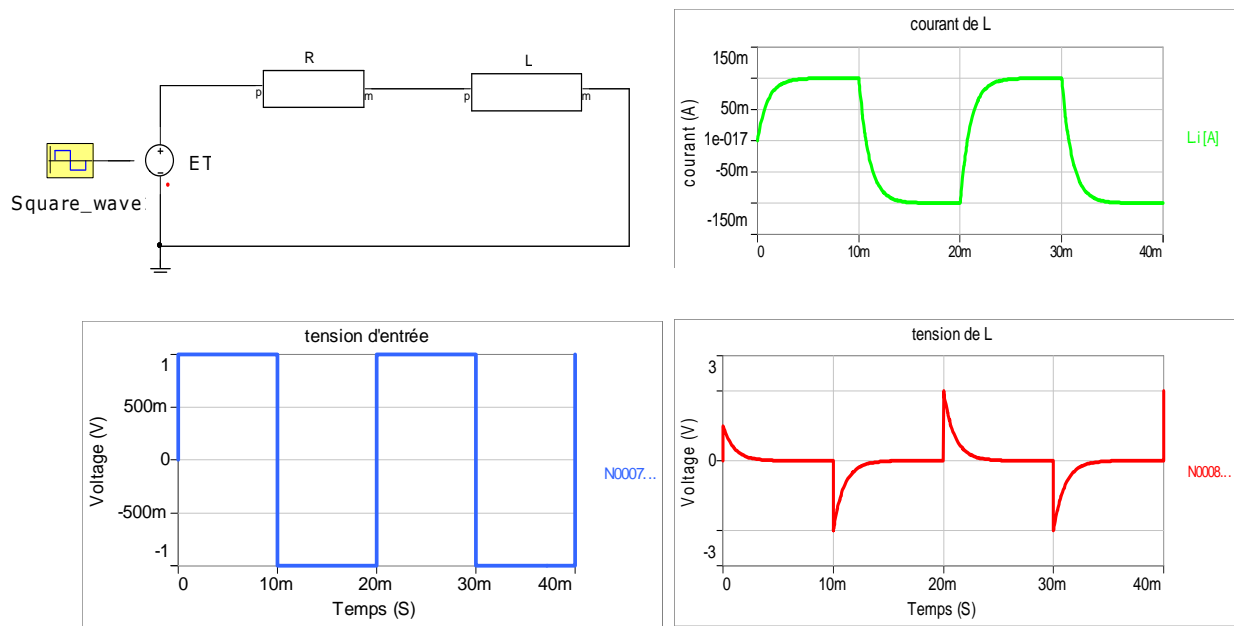


Figure III.7: Modélisation et simulation par Simplorer d'un circuit RL.

De cette manière, l'utilisateur non spécialiste des langages pourra simplement agencer des boîtes graphiques les unes avec les autres de manière à représenter au mieux la fonction du système qu'il souhaite simuler sans avoir à établir le code correspondant (Figure III.8).

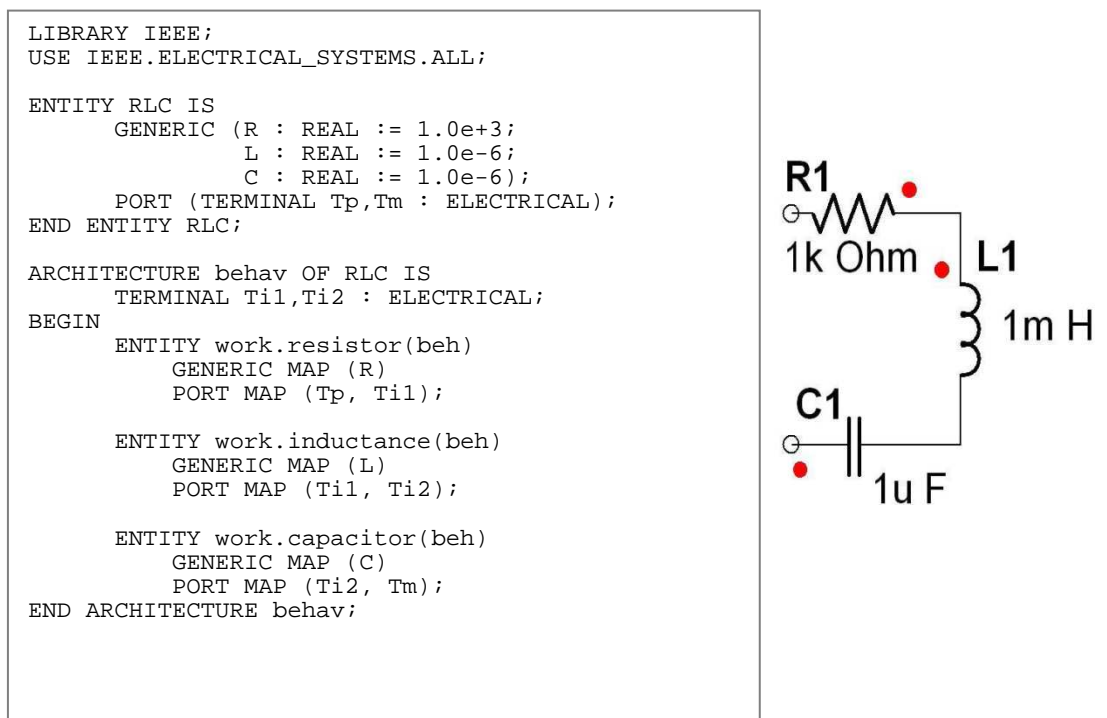


Figure III.8: Modèle VHDL-AMS et représentation graphique d'un circuit RLC série (Simplorer 7.0).

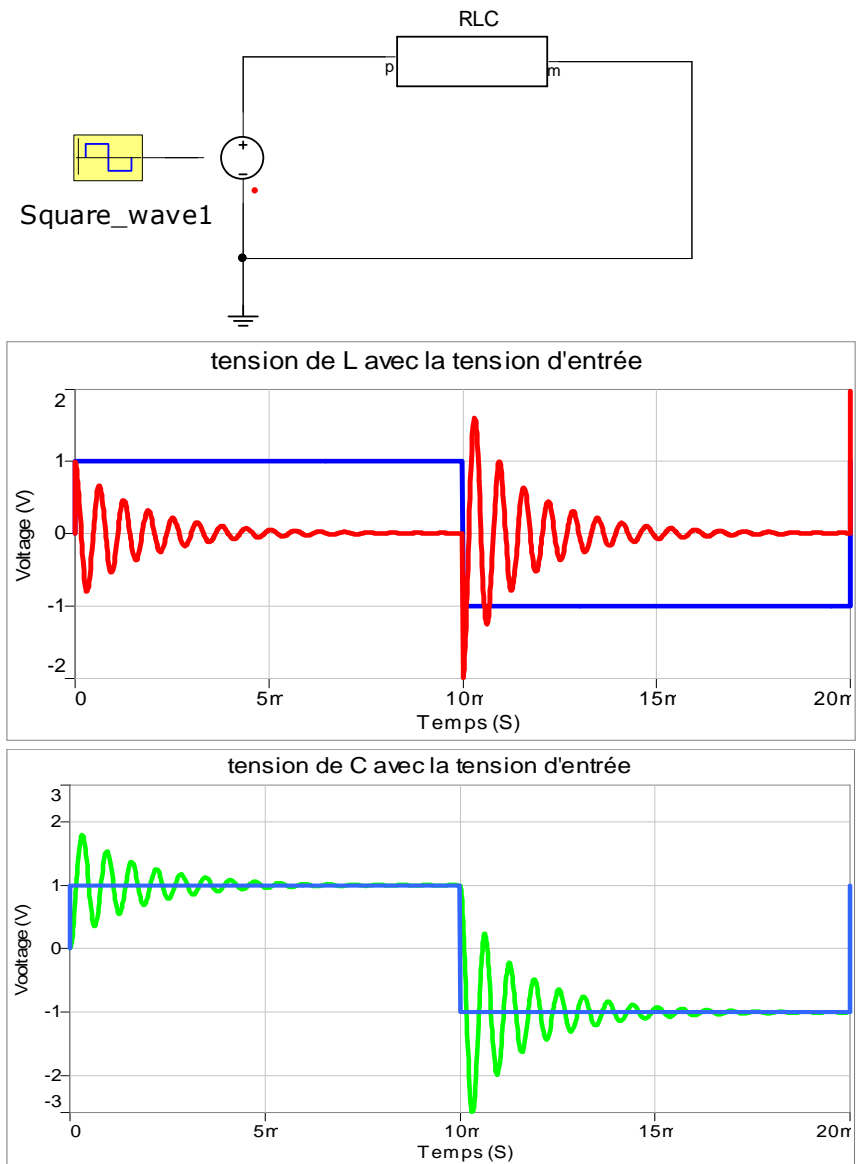


Figure III.9: Modélisation et simulation par Simplorer d'un circuit RLC.

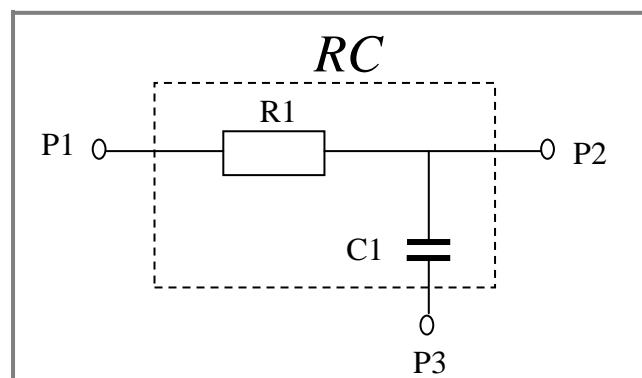


Figure III.10: Hiérarchie simple d'un circuit RC.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE WORK.ALL;

ENTITY RC IS
  PORT (TERMINAL P1, P2, P3 : ELECTRICAL);
END ENTITY RC;

ARCHITECTURE behav OF RC IS
BEGIN
  R1: ENTITY R(behav)
    GENERIC MAP (R=> 10.0E+3)
    PORT MAP (P1, P2);
  C1: ENTITY C(behav)
    GENERIC MAP (C=>1.0E-6)
    PORT MAP (P2, P3);
END ARCHITECTURE behav;

```

Figure III.11: Modèle VHDL-AMS d'un filtre RC.

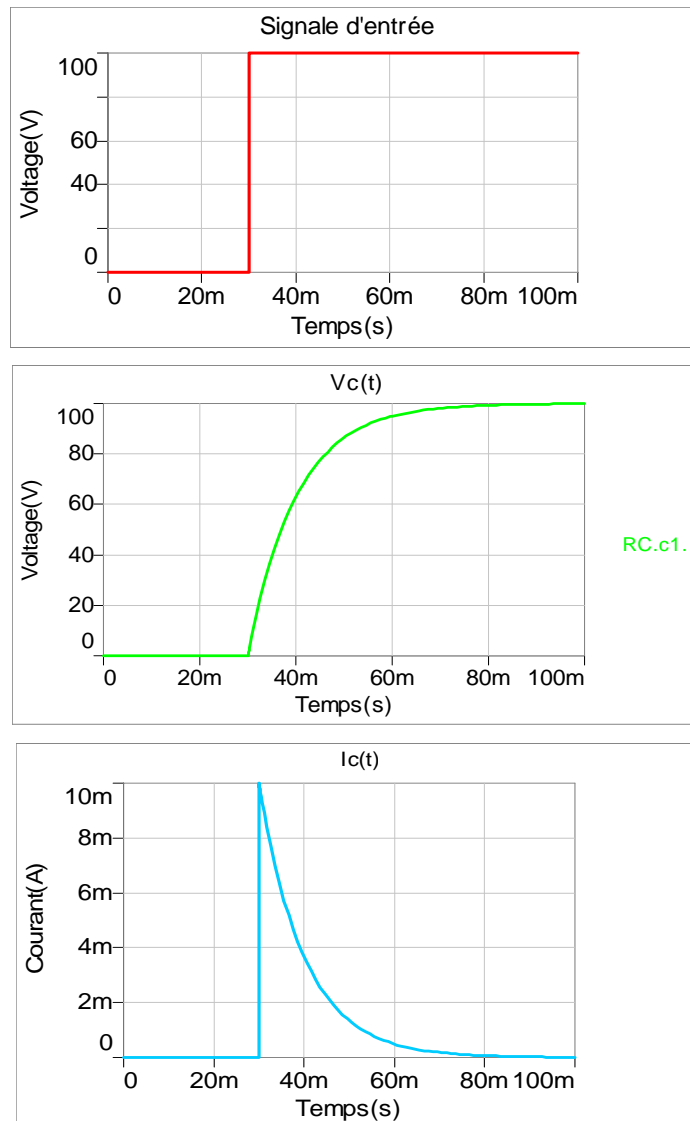


Figure III.12: Résultats de simulation d'un filtre.

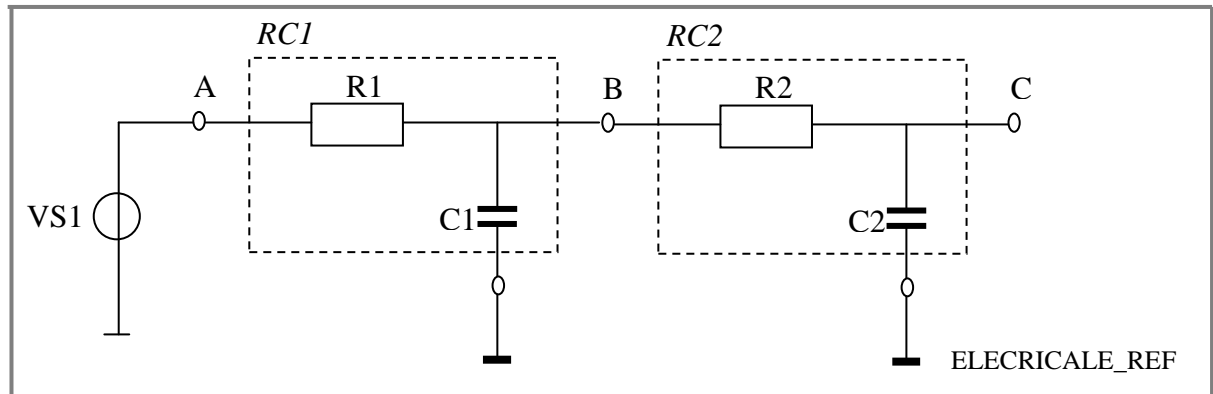


Figure III.13: Test-bench d'un circuit double RC [10].

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE WORK.ALL;

ENTITY D_RC IS
END ENTITY D_RC;

ARCHITECTURE behav OF D_RC IS
    TERMINAL A, B, C : ELECTRICAL;
    BEGIN

        VS1: ENTITY STEP(V1)
            GENERIC MAP (T_RISE => 2.0E-3)
            PORT MAP (A, ELECTRICAL_REF);

        RC1: ENTITY RC(V1)
            GENERIC MAP (4.0E2, 1.0E-6)
            PORT MAP (A, B, ELECTRICAL_REF);

        RC2: ENTITY RC(V1)
            GENERIC MAP (1.0E3, 1.0E-6)
            PORTMAP (B, C, ELECTRICAL_REF);
    END ARCHITECTURE behav;

```

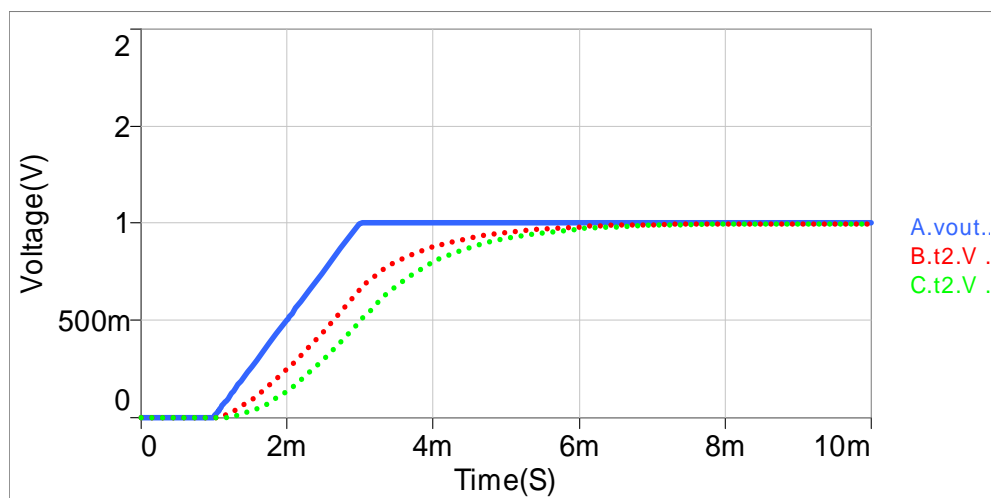


Figure III.14: Modèle VHDL-AMS et résultats de simulation d'un circuit double RC.

III.3.4 Diode

Il y a deux choix de modèle d'une diode : soit en mode d'un interrupteur (idéal) soit en mode non linéaire.

L'équation suivant représente le courant I_d d'une diode

$$I = I_{SAT} * (\exp (v/VT) - 1.0) \quad (III.5)$$

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY diode IS
    GENERIC (r_on : REAL := 1.0E-3 ; r_of : REAL := 1.0E+5 );
    PORT (TERMINAL A,C :ELECTRICAL);
END ENTITY diode;

ARCHITECTURE behav OF DIODE IS
    QUANTITY Vd ACROSS Id THROUGH A TO C;
    QUANTITY R : REAL ;
BEGIN
    IF Vd'ABOVE(0.00) USE
        R == r_on;
    ELSE
        R == r_of;
    END USE;
    Vd == R*Id;
END ARCHITECTURE behav;

```

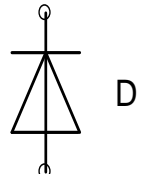


Figure III.15: Modèle VHDL-AMS et représentation graphique d'une diode (Simplorer 7.0).

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY D IS
    GENERIC (Isat : CURRENT := 1.0e-12;
            VT : VOLTAGE := 26.0e-3;
            Rr : RESISTANCE := 100.0e3);
    PORT (TERMINAL A,C : ELECTRICAL);
END ENTITY D;

ARCHITECTURE behav OF D IS
    QUANTITY V ACROSS I THROUGH A TO C;
BEGIN
    IF (v >= 0.0) USE
        I == Isat * ((exp(V/VT)) - 1.0);
    ELSE
        I == V/Rr;
    END USE;
END ARCHITECTURE behav;

```

Figure III.16: Modèle VHDL-AMS d'une diode non linéaire.

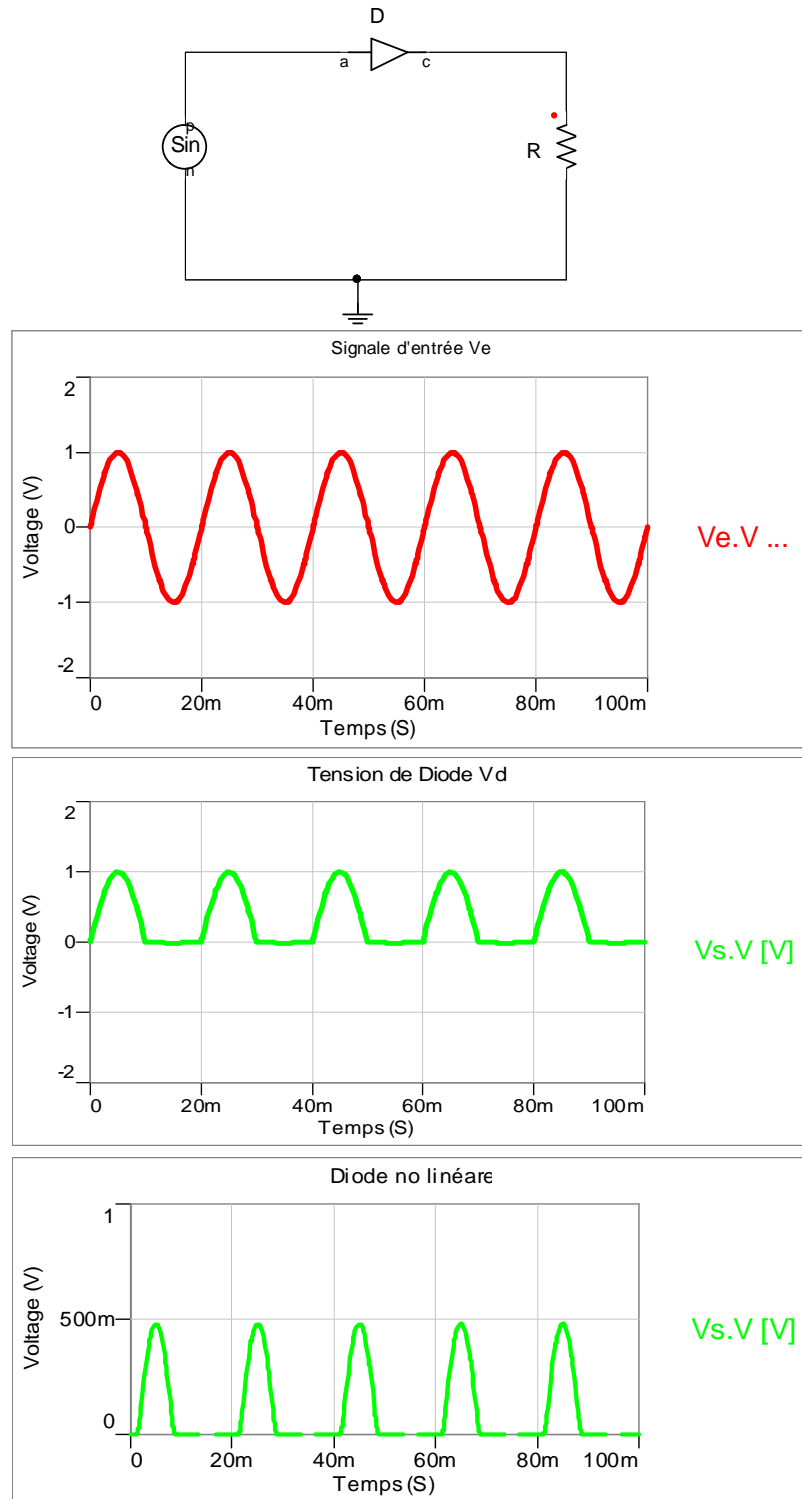


Figure III.17: Modélisation et simulation par Simplorer d'une diode idéal et non linéaire.

III.4 Connexion mathématique

Les modèles de haut-niveau sont définis comme des blocs fonctionnels utilisés par exemple dans des schémas-blocs pour étudier la stabilité de systèmes de régulation électrique ou mécanique ou autre. Les grandeurs analogiques manipulées sont donc plutôt considérées comme des variables mathématiques et il n'y a pas lieu de soumettre les connexions entre

blocs aux lois de Kirchhoff. Signalons que de telles connexions sont aussi établies par les quatre sources contrôlées idéales du simulateur SPICE qui permettent de lire soit des tensions en des nœuds de contrôle, soit des courants circulant à travers des sources de tension ou des résistances, sans introduire de perturbations.

Il est donc nécessaire de disposer d'un autre type de connexion, semblable au ***couplage*** (***coupling***) de flux entre deux inductances mutuelles (Figure III.18): le flux magnétique de chaque inductance couplée dépend à la fois du courant qui circule dans ses spires et du courant circulant dans l'autre inductance. Le modèle d'une inductance couplée doit donc avoir trois points de connexion: deux bornes électriques et un couplage.

La connexion de couplage possède d'autre part une direction: la variable transmise par ce biais est exportée d'un modèle qui en définit la valeur et importée dans un modèle qui ne fait que lire cette valeur [4].

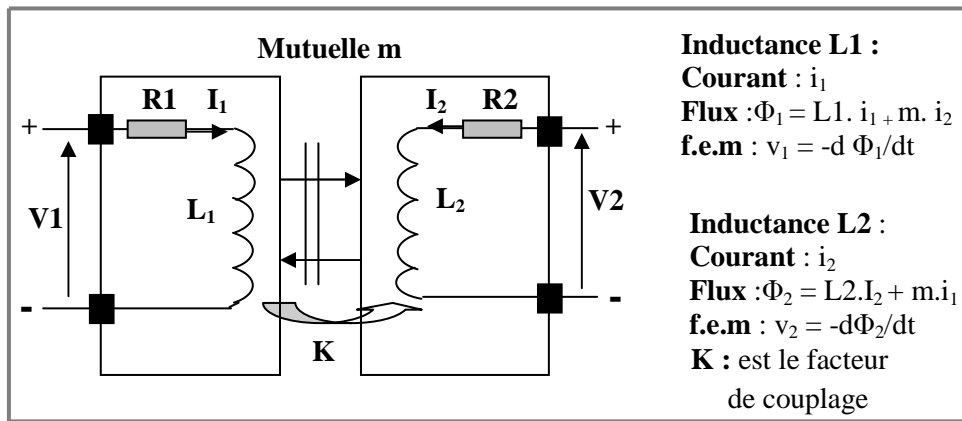


Figure III.18: Équations d'inductances couplées $m = K\sqrt{L_1 \cdot L_2}$

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY transf IS
  GENERIC (L1 : REAL := 1.0E-6;
           R1 : REAL := 0.0;
           L2 : REAL := 1.0E-6;
           R2 : REAL := 0.0;
           K : REAL := 0.5);
  PORT (TERMINAL P1, P0, S1, S0 : ELECTRICAL);
END transf;

ARCHITECTURE behav OF transf IS
  QUANTITY M : REAL := K * SQRT(L1*L2);
  QUANTITY V1 ACROSS I1 THROUGH P1 TO P0;
  QUANTITY V2 ACROSS I2 THROUGH S1 TO S0;
BEGIN
  M == K * SQRT(L1*L2);
  V1 == L1 * I1'DOT + M * I2'DOT + R1 * I1;
  V2 == M * I1'DOT + L2 * I2'DOT + R2 * I2;
END behav;

```

Figure III.19: Modèle VHDL-AMS d'un transformateur composé par deux bobines.

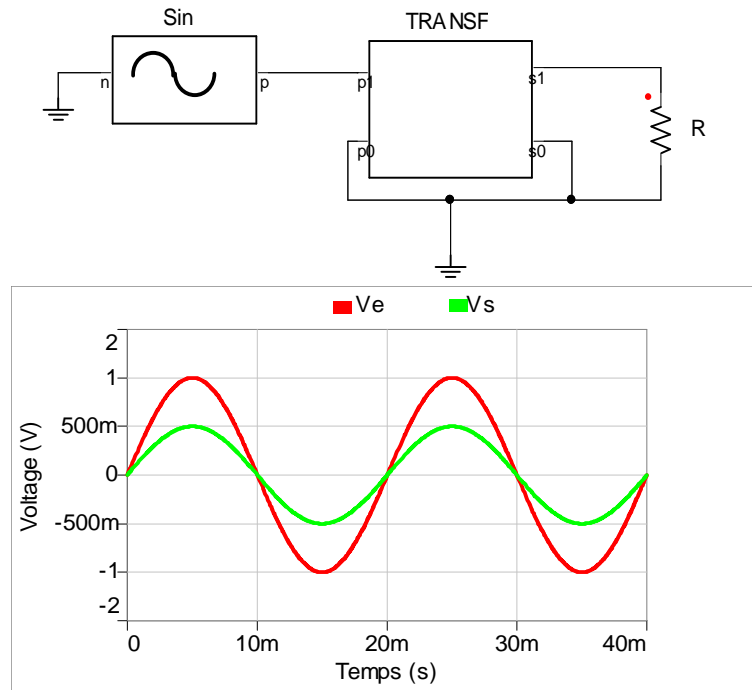


Figure III.20: Modélisation et simulation d'un transformateur composé par deux bobines.

III.5 Générateurs

III.5.1 Sources indépendantes constantes

III.5.1.1 Source de tension indépendante constante

Le programme VHDL-AMS de la figure III.21 donne le modèle d'une source de tension indépendante constante. Il faut noter que la quantité de branche `Iout` doit être déclarée, même si elle n'est pas explicitement référencée dans l'équation constitutive de la source. La raison en est que seule une quantité de branche de type **through** crée une branche topologique entre deux terminaux. D'autre part, le courant `Iout` existe physiquement et sa valeur sera définie par la charge appliquée à la source. Finalement, le modèle est idéal et une source de tension réelle possède une résistance interne non nulle qui n'est pas représentée ici [2].

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY source_v IS
  GENERIC (V0: REAL);
  PORT (TERMINAL P, M: ELECTRICAL);
END ENTITY source_v ;

ARCHITECTURE behav OF source_v IS
  QUANTITY VOUT ACROSS IOUT THROUGH P TO M;
BEGIN
  VOUT == V0;
END ARCHITECTURE behav;

```

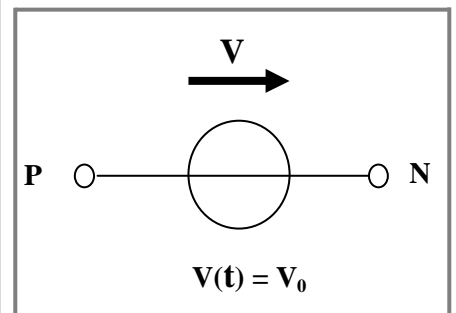


Figure III.21: Modèle VHDL-AMS et schéma électrique de source de tension indépendante constante.

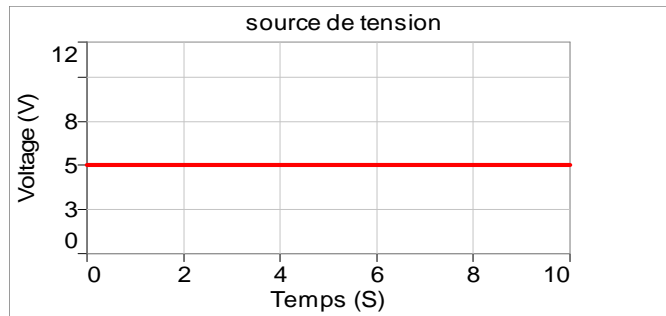


Figure III.22: Résultat de simulation de source de tension indépendante constante.

III.5.1.2 Source de courant indépendante constante

Le programme VHDL-AMS de la figure III.23 donne le modèle d'une source de courant indépendante constante. Dans ce cas, la déclaration de la quantité de branche Vout est optionnelle.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY source_i IS
    GENERIC (I0: REAL);
    PORT (TERMINAL P, M: ELECTRICAL);
END ENTITY source_i;

ARCHITECTURE behav OF source_i IS
    QUANTITY Vout ACROSS Iout THROUGH P TO M;
BEGIN
    Iout == I0;
END ARCHITECTURE behav;

```

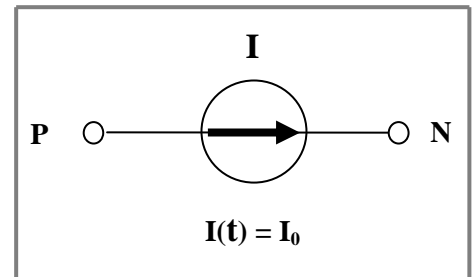


Figure III.23: Modèle VHDL-AMS et schéma électrique de source de courant indépendante constante.

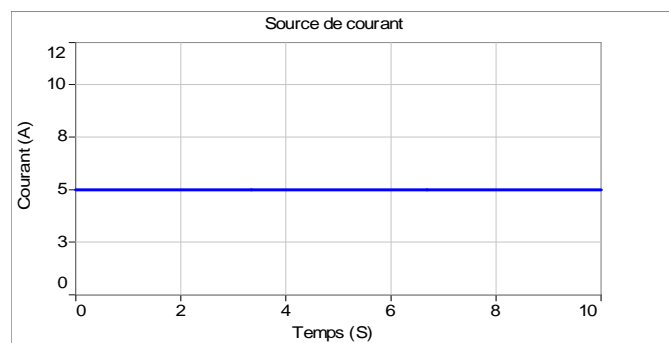


Figure III.24: Résultat de simulation de source de courant indépendante.

III.5.2 Source de tension Sinusoïdal

L'équation d'une source de tension sinusoïdale est donnée par l'équation (III.6).

$$V(t) = V_0 + A * \sin(2\pi ft) \quad (\text{III.6})$$

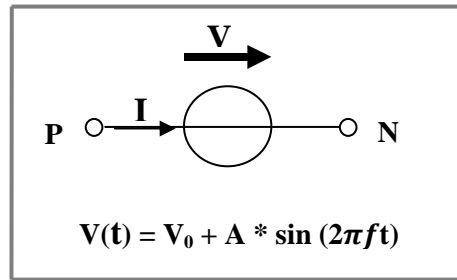


Figure III.25: Source de tension Sinusoïdal.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY sine IS
  GENERIC (V0 : REAL := 0.0;
           A : REAL := 1.0;
           FREQ : REAL := 50.0);
  PORT (TERMINAL P, N : ELECTRICAL);
END ENTITY sine;

ARCHITECTURE behav OF sine IS
  QUANTITY V ACROSS I THROUGH P TO N;
BEGIN
  V == V0 + A*SIN(MATH_2_PI*FREQ*NOW);
END ARCHITECTURE behav;

```

Figure III.26: Modèle VHDL-AMS de source de tension sinusoïdale.

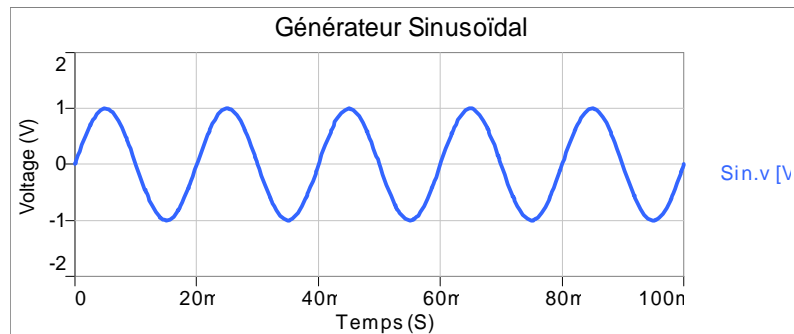


Figure III.27: Résultat de simulation de source de Sinusoïdal.

III.5.3 Générateur d'impulsion

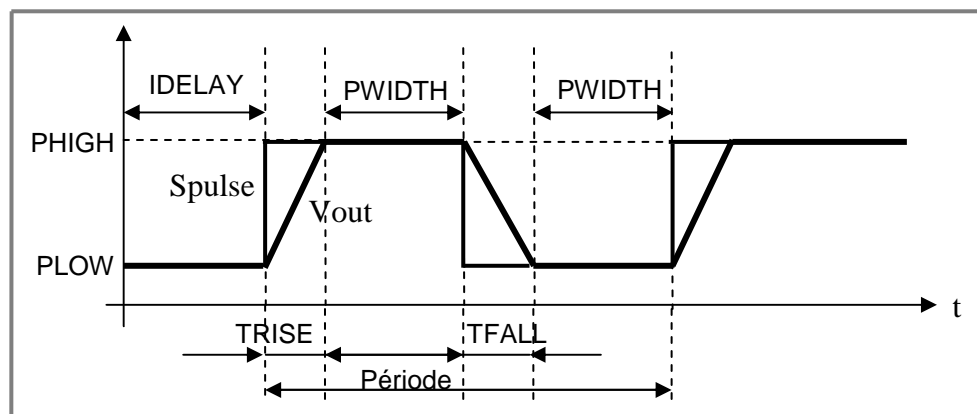


Figure III.28: Paramètres du générateur d'impulsion [2].

Dans la figure III.29, un processus est responsable de la génération d'un signal de type réel, spulse, qui définit les instants de changements pour la quantité qui va représenter le train d'impulsions. La boucle interne au processus va impliquer une répétition avec une période égale à : $TRISE + TFALL + 2*PWIDTH$. L'instruction simultanée qui suit le processus utilise la quantité implicite S' ramp pour "lisser" la forme d'onde générée et la rendre continue (ou plutôt linéaire par morceaux).

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY pulse IS
    GENERIC (PLOW : REAL := 0.0; -- Valeur basse
             PHIGH: REAL := 5.0; -- Valeur haute
             TRISE: REAL := 1.0E-6; -- Temps de montée
             TFALL: REAL := 1.0E-6; -- Temps de descente
             IDELAY: TIME := 1 MS; -- Délai initial
             PWIDTH: TIME := 5 MS); -- Largeur de l'impulsion
    PORT (TERMINAL P,M: ELECTRICAL);
END ENTITY pulse;

ARCHITECTURE behav OF pulse IS
    QUANTITY Vout ACROSS IOUT THROUGH P TO M;
    SIGNALSPULSE: REAL := PLOW;
BEGIN
    PROCESS
    BEGIN
        WAIT FOR IDELAY; -- Délai initial
        LOOP
            SPULSE<= PHIGH;
            WAIT FOR PWIDTH;
            SPULSE<= PLOW;
            WAIT FOR PWIDTH;
        END LOOP;
    END PROCESS;
    Vout == SPULSE'RAMP(TRISE, TFALL);
END ARCHITECTURE behav;

```

Figure III.29: Modèle VHDL-AMS d'un générateur d'impulsion.

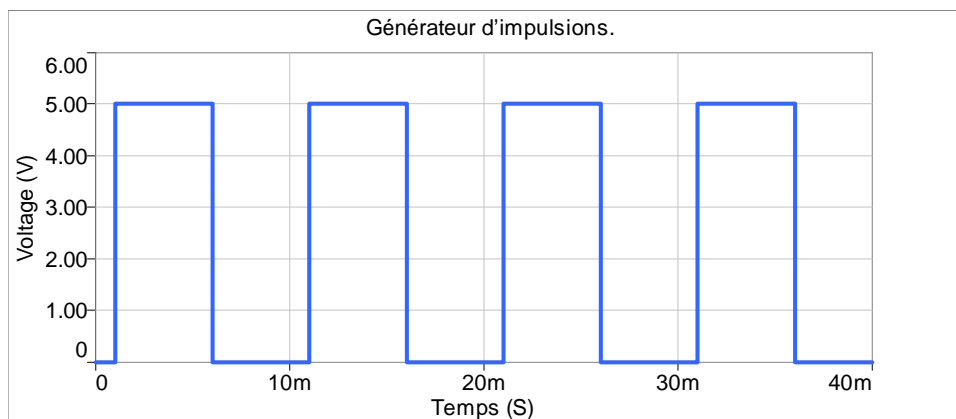


Figure III.30: Résultat de simulation d'un générateur d'impulsion.

III.6 Trigger de Schmitt

C'est un commutateur à hystérésis, il bascule lorsque la tension à son entrée dépasse un seuil V_0 et revient dans son état initial si cette tension descend en dessous de $V_1 < V_0$. Il est construit classiquement avec un amplificateur opérationnel ou deux transistors, mais la modélisation comportementale permet de le réaliser avec un nombre réduit de nœuds. Ce qui est particulièrement important lorsque l'on utilise la version d'évaluation du logiciel.

III.6.1 Trigger inverseur

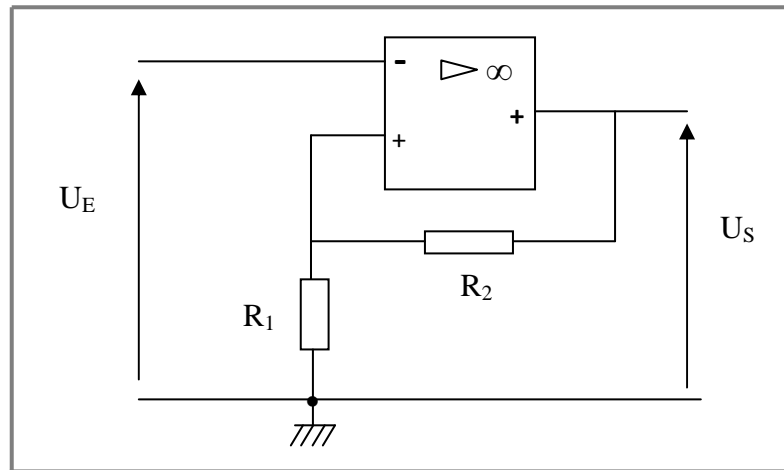


Figure III.31: Montage d'un trigger de schmitt inverseur.

Théorème de Millman :

$$U_S = \frac{R_1}{R_1 + R_2} U_E \quad (\text{III.7})$$

$$\text{Si } U_S = V_{\text{sat}^-} \Rightarrow U_E = U_B = \frac{R_1}{R_1 + R_2} V_{\text{sat}^-} \quad (\text{III.8})$$

$$\text{Si } U_S = V_{\text{sat}^+} \Rightarrow U_E = U_H = \frac{R_1}{R_1 + R_2} V_{\text{sat}^+} \quad (\text{III.9})$$

A.N

$V_{\text{cc}\pm} = \pm 15 \text{ V}$

$R_1 = 1 \text{ k}\Omega$, $R_2 = 2 \text{ k}\Omega$

$U_H = 1/3 * (15) = +5\text{V}$

$U_B = 1/3 * (-15) = -5\text{V}$

Le schéma de la figure III.32 représente la simulation d'un trigger de schmitt inverseur par PSPICE.

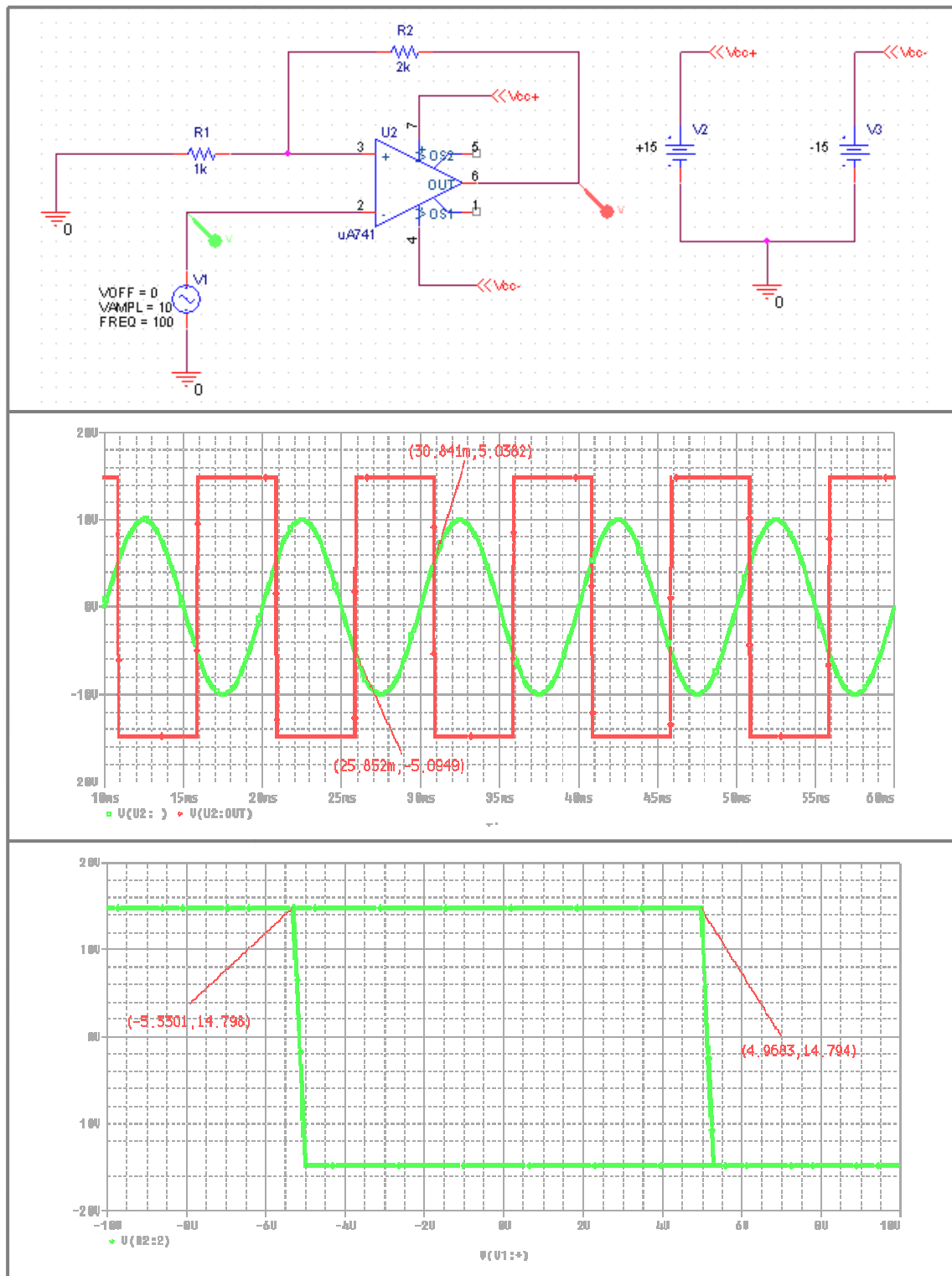


Figure III.32: Résultats de simulation d'un trigger de schmitt inverseur par PSPICE.

III.6.1.1 Modèle PSPICE du Trigger inverseur

Le schéma de la figure III.33 représente la modélisation comportementale d'un trigger de schmitt.

Dans cet exemple ci-dessous l'ampli op est modélisé par un bloc ABM2 effectuant l'opération $(V_{in+} - V_{in-}) \cdot 100K$ suivi d'un limiteur $\pm 15V$. Cet ampli est ensuite bouclé sur l'entrée + par une résistance de $2K$. Avec une résistance en série sur l'entrée de $1K$ les deux seuils sont $\pm 5V$.

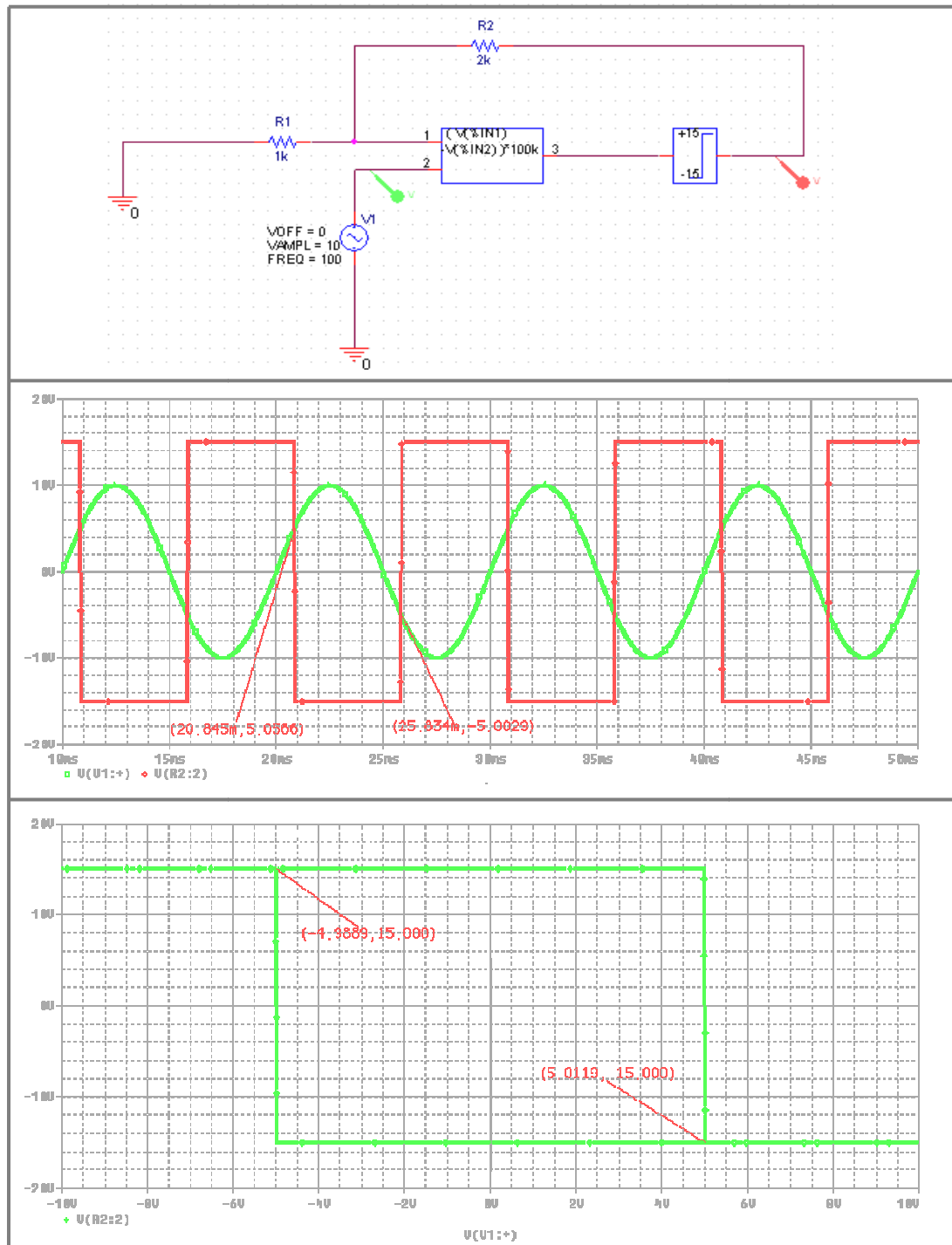


Figure III.33: Résultats de modélisation et simulation d'un trigger de schmitt inverseur par PSPICE.

III.6.1.2 Modèle VHDL-AMS du Trigger inverseur

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY trigger_schmitt IS
END ENTITY trigger_schmitt;

ARCHITECTURE hysteresis OF trigger_schmitt IS
    SIGNAL STATE : REAL := 0.0;
    SIGNAL A1,A2 : BOOLEAN := FALSE;
    QUANTITY VIN,VOUT : REAL;
BEGIN
    A1 <= VIN'ABOVE(5.0);
    A2 <= VIN'ABOVE(-5.0);
    STATE <= -15.0 WHEN A1
    ELSE
    15.0 WHEN NOT A2
    ELSE UNAFFECTED;
    Vin == 10.0 * SIN(2.0 * MATH_PI*0.5E3*NOW);
    Vout == STATE'RAMP(1.0E-9,1.0E-9);
END ARCHITECTURE hysteresis;

```

Figure III.34: Modèle VHDL-AMS du Trigger inverseur.

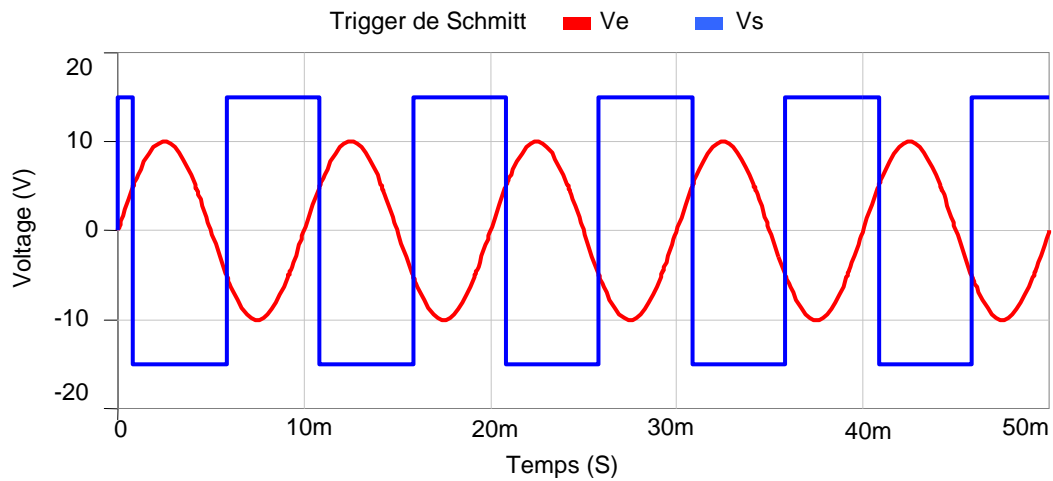


Figure III.35: Résultats de simulation d'un trigger de schmitt inverseur.

III.6.2 Trigger non inverseur

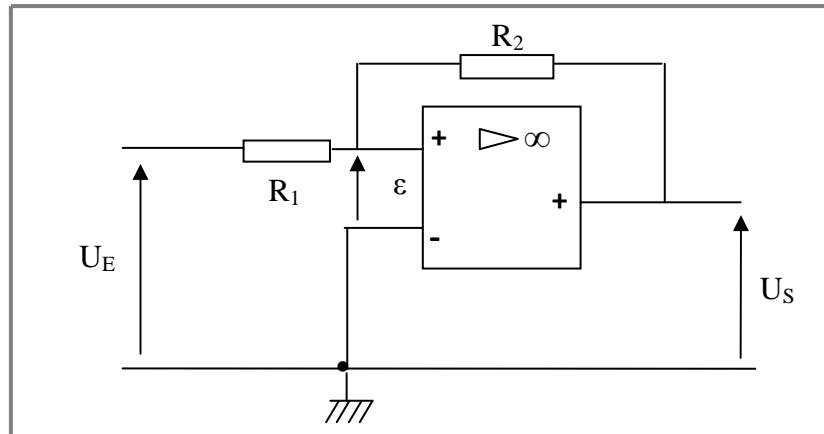


Figure III.36: Montage d'un trigger de schmitt non inverseur.

- Calcul des tensions de seuil

A l'instant du basculement de la sortie : $\epsilon = 0 \text{ V} \Rightarrow V_+ = 0$

Théorème de Millman :

$$V_+ = \frac{\frac{U_E}{R_1} + \frac{U_S}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2}} \Rightarrow U_E = -\frac{R_1}{R_2} U_S \quad (\text{III.10})$$

$$\text{Si } U_S = V_{\text{sat}}^- \Rightarrow U_E = U_H = -\frac{R_1}{R_2} V_{\text{sat}}^- \quad (\text{III.11})$$

$$\text{Si } U_S = V_{\text{sat}}^+ \Rightarrow U_E = U_B = -\frac{R_1}{R_2} V_{\text{sat}}^+ \quad (\text{III.12})$$

A.N

$V_{\text{cc}\pm} = \pm 15 \text{ V}$

$R_1 = 1 \text{ k}\Omega$, $R_2 = 2 \text{ k}\Omega$

$$U_H = -1/2 * (15) = -7.5 \text{ V}$$

$$U_B = 1/2 * (15) = +7.5 \text{ V}$$

Le schéma de la figure III.37 représente les résultats de simulation d'un trigger de schmitt non inverseur par PSPICE.

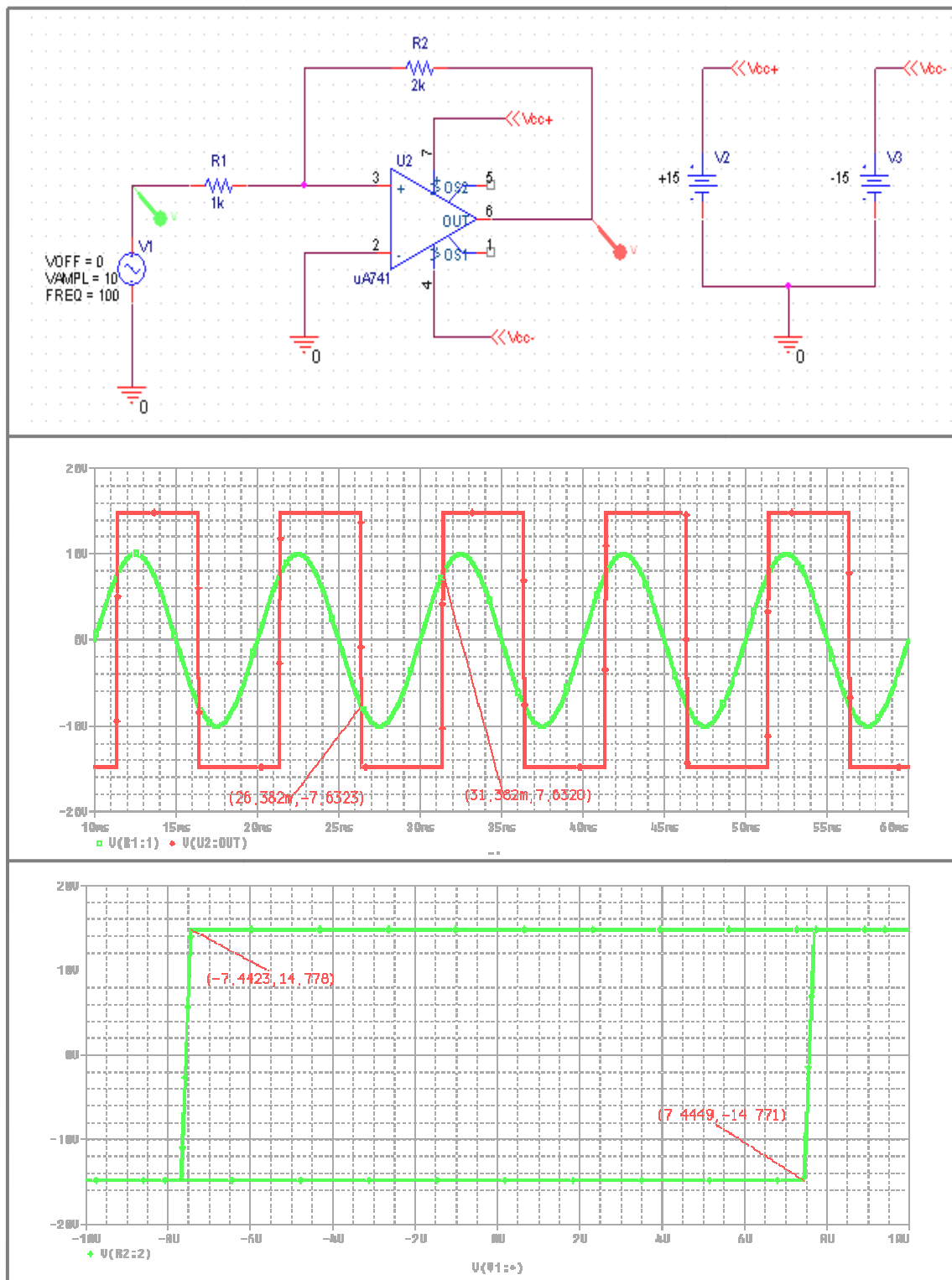


Figure III.37: Résultats de simulation d'un trigger de schmitt non inverseur par PSPICE.

La simulation montre le cycle d'Hystérésis du trigger de schmitt inverseur.

III.6.2.1 Modèle PSPICE de trigger non inverseur

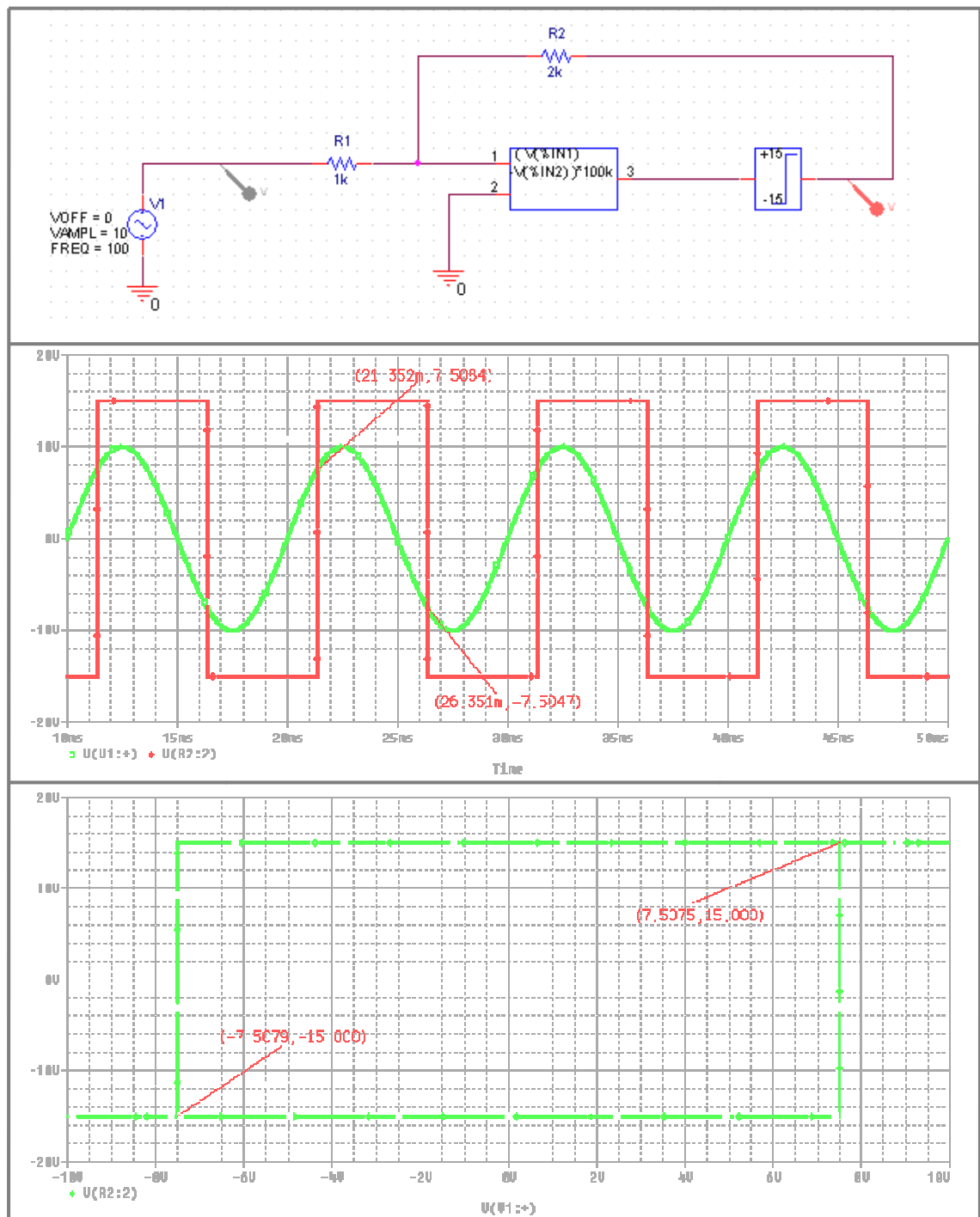


Figure III.38: Résultats de modélisation et simulation d'un trigger de schmitt non inverseur par PSPICE.

Le cycle d'Hystérésis non inversé est reproduit fidèlement par le macromodèle que nous avons conçu.

III.6.2.2 Modèle VHDL-AMS de Trigger non inverseur

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY trigger_schmitt IS
END ENTITY trigger_schmitt;

ARCHITECTURE hysteresis OF trigger_schmitt IS
    SIGNAL STATE : REAL := 0.0;
    SIGNAL A1,A2 : BOOLEAN := FALSE;
    QUANTITY VIN,VOUT : REAL;
BEGIN
    A1 <= VIN'ABOVE(5.0);
    A2 <= VIN'ABOVE(-5.0);
    STATE <= 15.0 WHEN A1
    ELSE
    -15.0 WHEN NOT A2
    ELSE UNAFFECTED;
    VIN == 10.0 * SIN(2.0 * MATH_PI*1.0E2*NOW);
    VOUT == STATE'RAMP(1.0E-9,1.0E-9);
END ARCHITECTURE hysteresis;

```

Figure III.39: Modèle VHDL-AMS du Trigger non inverseur.

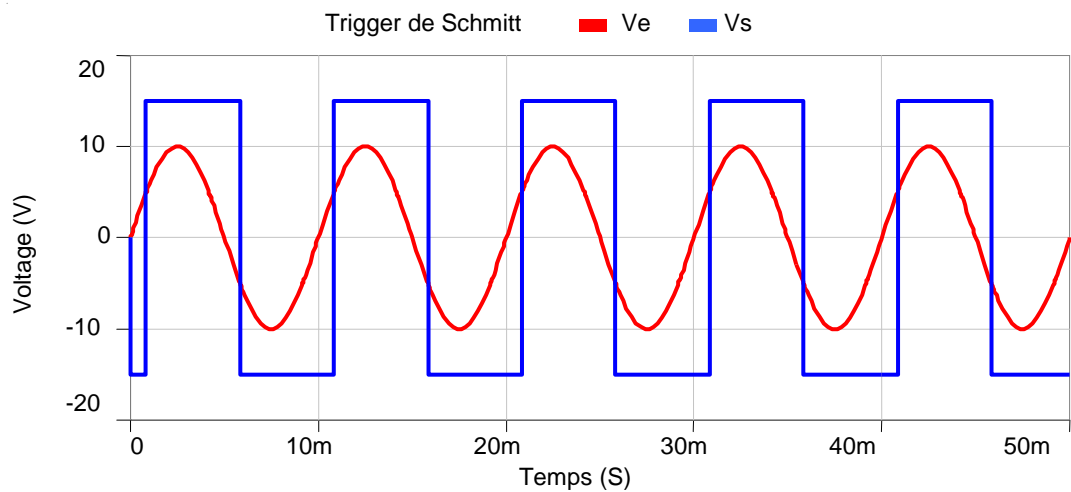


Figure III.40: Résultats de simulation d'un trigger de schmitt non inverseur (SIMPLORER V7).

L'écriture d'un programme VHDL-AMS pour un trigger de schmitt non inverseur donne des résultats identiques à ceux obtenues par macromodélisation.

III.7 Contrôle du système hybride

On modélisé le block de la figure III.41 par PSPICE et le VHDL-AMS

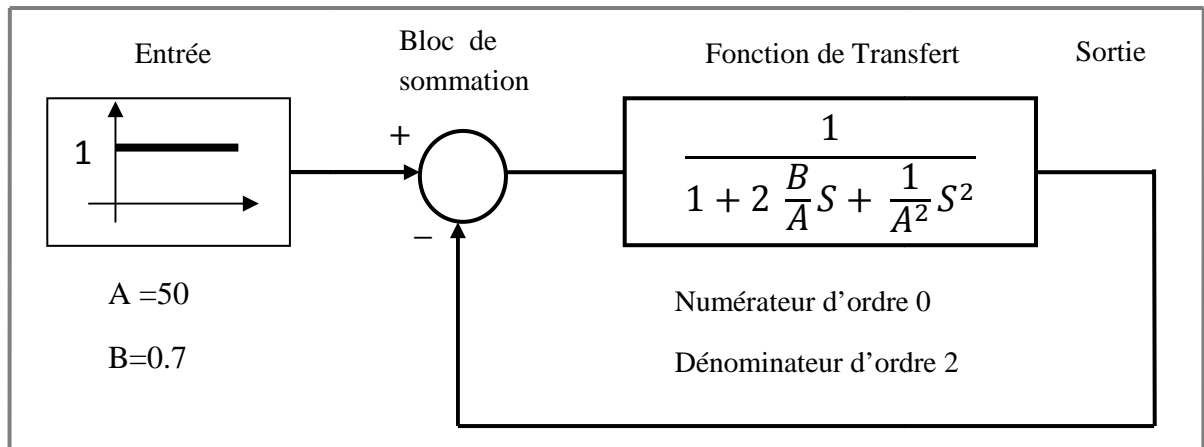


Figure III.41: Schéma du bloc fonctionnel.

III.7.1 Modélisation par PSPICE

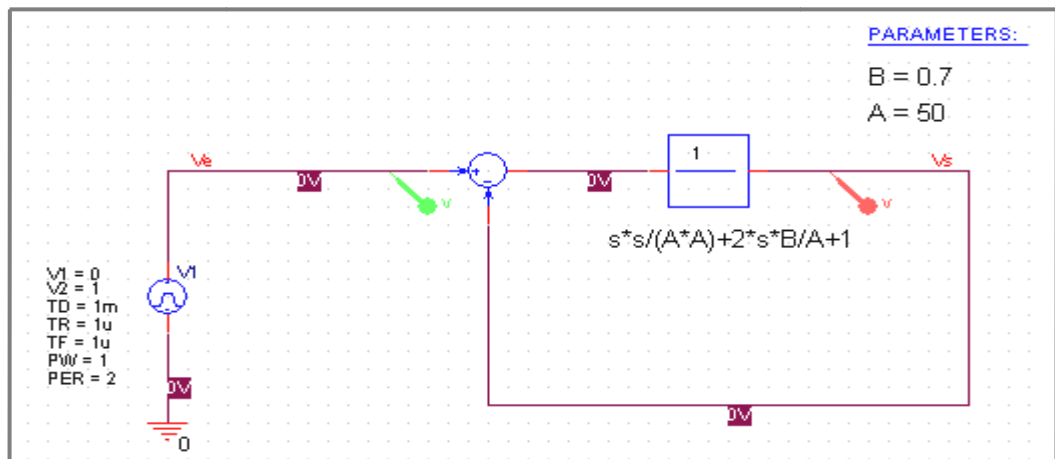


Figure III.42: Première méthode de modélisation par PSPICE.

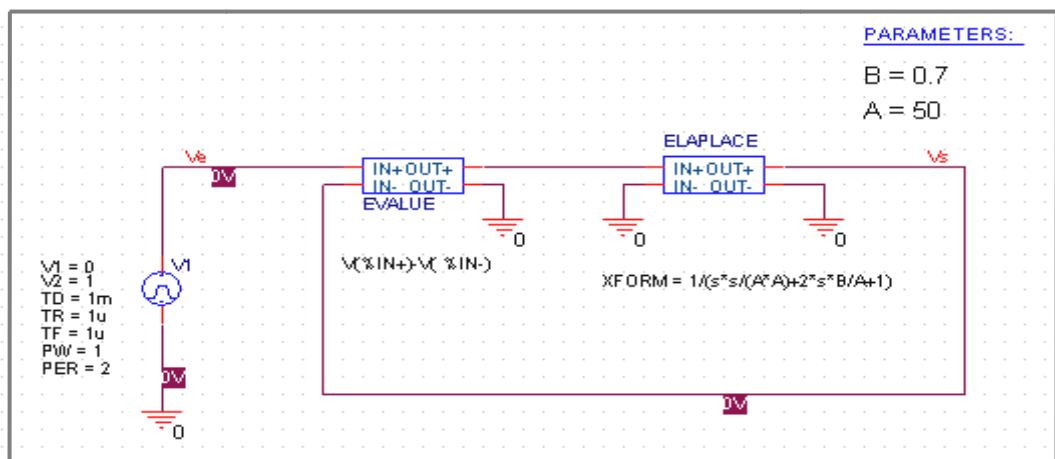


Figure III.43: Deuxième méthode de modélisation par PSPICE.

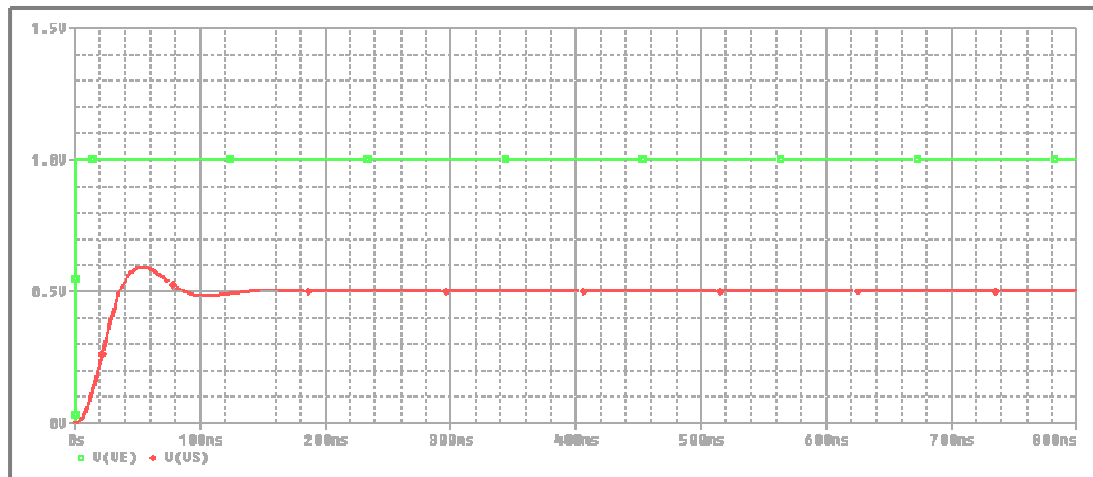


Figure III.44: Résultat de simulation pour les deux méthodes par PSPICE.

La sortie du bloc fonctionnel correspond à l'entrée après une courte durée de transition (régime transitoire), reste à noter que l'amplitude de la sortie est égale à un demi de l'entrée (c'est un choix)

III.7.1 Modélisation par VHDL-AMS

- **Sommateur**

```

ENTITY SUM IS
  PORT (QUANTITY INPUT0, INPUT1 : IN REAL;
        QUANTITY VAL : OUT REAL);
END ENTITY SUM;

ARCHITECTURE behav OF SUM IS
  QUANTITY temp_val : REAL := 0.0;
BEGIN
  temp_val == INPUT0 + INPUT1;
  VAL == temp_val;
END behav;

```

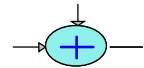


Figure III.45: Modèle VHDL-AMS et représentation graphique d'un sommateur (Simplorer 7.0).

- **Fonction de transfert**

```

ENTITY GS IS
  GENERIC(N : INTEGER := 1; D : INTEGER := 2;
          NUM : REAL_VECTOR(0 TO 0) := (OTHERS => 0.0);
          DEN : REAL_VECTOR(0 TO 2) := (OTHERS => 0.0));
  PORT (QUANTITY INPUT : IN REAL;
        QUANTITY VAL : OUT REAL);
END ENTITY GS;

ARCHITECTURE behav OF GS IS
BEGIN
  val == INPUT'LTF(NUM,DEN);
END ARCHITECTURE behav;

```

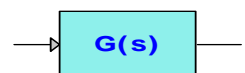


Figure III.46: Modèle VHDL-AMS et représentation graphique d'une fonction de transfert (Simplorer 7.0).

- Soustracteur

```

ENTITY NEG IS
PORT (QUANTITY INPUT : IN REAL;
      QUANTITY VAL : OUT REAL);
END ENTITY NEG;

ARCHITECTURE behav OF NEG IS
  QUANTITY temp_val : REAL := 0.0;
BEGIN
  temp_val == -INPUT;
  VAL == temp_val;
END ARCHITECTURE behav;

```



Figure III.47: Modèle VHDL-AMS et représentation graphique d'un soustracteur.

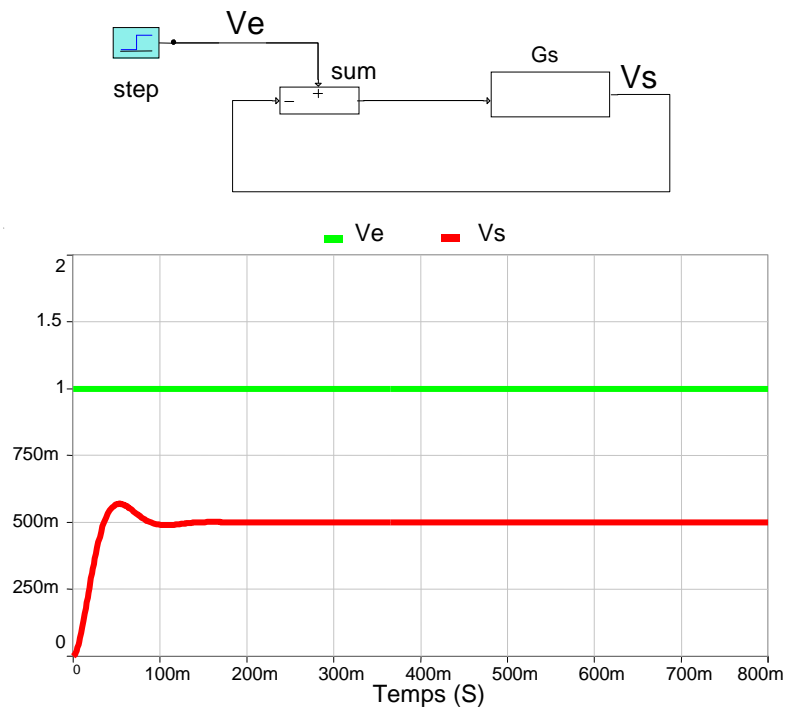


Figure III.48: Résultat de modélisation et simulation par SIMPLORER d'un bloc fonctionnel.

Le modèle VHDL-AMS pour un bloc fonctionnel donne des résultats identiques à ceux obtenues par macromodélisation.

III.8 Machine à courant continu

Une machine à courant continu est une machine électrique. Il s'agit d'un convertisseur électromécanique permettant la conversion bidirectionnelle d'énergie entre une installation électrique parcourue par un courant continu et un dispositif mécanique. Elle est aussi appelée dynamo.

III.8.1 Modèle électrique

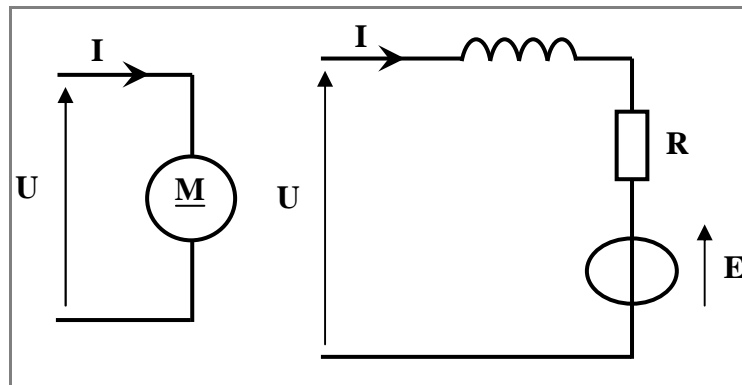


Figure III.49: Modèle électrique d'une machine à courant continu.

Nous avons ici le modèle simplifié d'un moteur à courant continu.

R : Résistance de l'induit (en ohm)

E : Force électromotrice (en Volt)

U : Tension appliquée aux bornes de l'induit.

$$\text{Force électromotrice : } E = k \cdot n \cdot \Phi \quad (\text{III.13})$$

(E : en volt, n : vitesse de rotation en tour/s (Hz), k : constante du moteur, Φ : flux magnétique)

$$\text{Loi d'ohm : } U = R \cdot I + E + L \frac{dI}{dt} \quad (\text{III.14})$$

$$\text{D'où Vitesse de rotation } n = \left(u - RI - L \frac{dI}{dt} \right) / (k\Phi) \quad (\text{III.15})$$

III.8.2 Modélisation par PSPICE

SPICE a été conçu pour simuler des circuits intégrés analogiques. Il ne possède donc bien évidemment pas de la possibilité de simuler les grandeurs mécaniques. Il est cependant possible de le faire en appliquant une analogie entre grandeurs mécaniques et grandeurs électriques. La correspondance choisie est :

Vitesse angulaire (rad/s) \Leftrightarrow Tension (V)

Couple (m.N) \Leftrightarrow Courant (A)

Les équations principales pour un moteur à courant continu :

✚ Equation électriques

La tension d'induit est :

$$U(t) = R \cdot i(t) + L \frac{di(t)}{dt} + e(t) \quad (\text{III.16})$$

$$e(t) = K\phi\Omega \quad (\text{III.17})$$

R : résistance d'induit,


L : inductance d'induit en henry,

K : constante tenant compte du flux constant,

e(t) : force électromotrice induite,

ϕ : flux total dans la machine,

Ω : vitesse angulaire du rotor (rad.s^{-1}).

 Equation mécaniques

Le principe fondamental de la dynamique (PFD) nous permet d'écrire

$$C_u - C_r = J \frac{d\Omega}{dt} \quad (\text{III.18})$$

$$C_u = C_{em} - C_p \quad (\text{III.19})$$

$$C_{em} = K\phi i \quad (\text{III.20})$$

$$C_r = f\Omega \quad (\text{III.21})$$

J : moment d'inertie de l'axe du rotor en Kg.m^2 ,

C_r : couple résistant,

C_{em} : couple électromagnétique,

C_u : couple utile,

C_p : couple de perte,

f : coefficient de frottement visqueux.

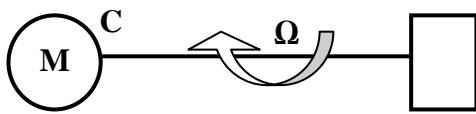
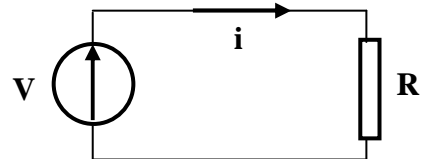
Elément mécanique	Elément électrique
Un moteur et une charge reliés par un axe rigide (Ω est commun) :	Dipôles en parallèle (v est commun)
	
Moment d'inertie : $J \frac{d\Omega}{dt} = \sum C$	Condensateur : $C \frac{dv}{dt} = i$
Couple visqueux : $C = K \cdot \Omega$	Conductance : $i = \frac{1}{R} v$

Tableau III.1: Comparaison entre les éléments mécanique et électronique.

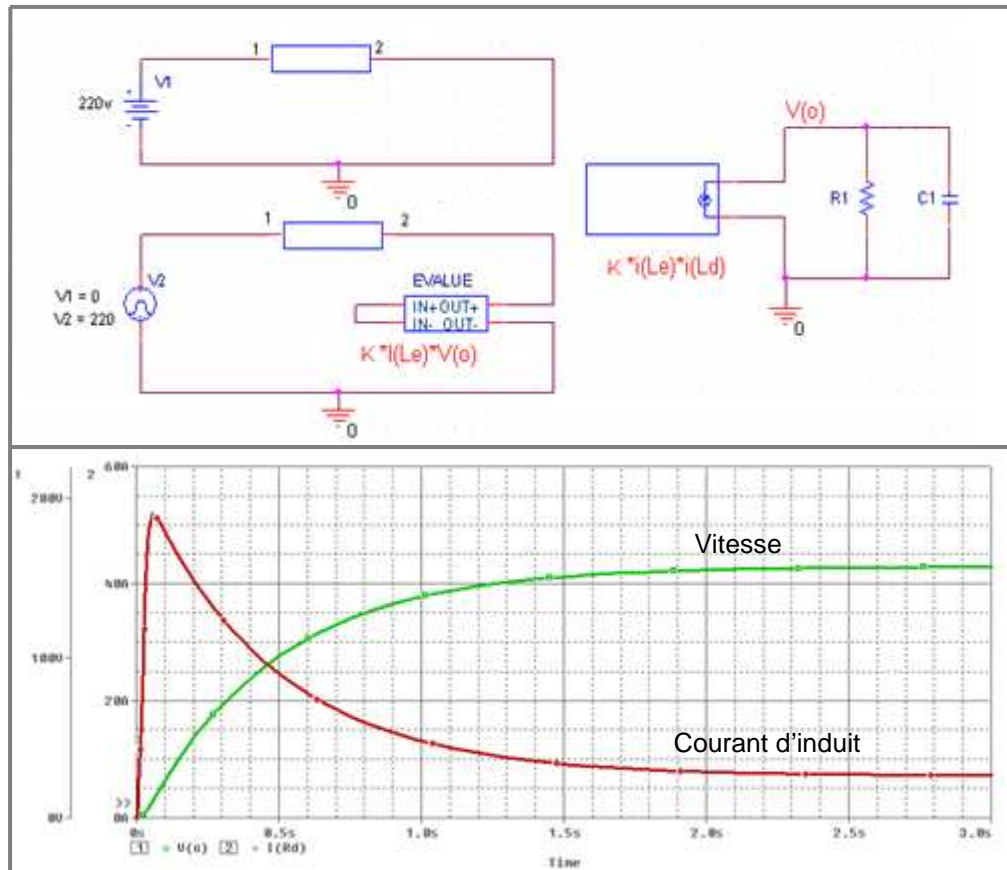


Figure III.50: Résultats de modélisation et simulation d'une machine à courant continu par PSPICE.

III.8.3 Modélisation par VHDL-AMS

```

LIBRARY IEEE;
USE IEEE.MATH_REAL.ALL;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MECHANICAL_SYSTEMS.ALL;

ENTITY motor IS
    GENERIC(
        RA : RESISTANCE := 1.0;
        LA : INDUCTANCE := 1.0E-3;
        KE : REAL := 1.0;
        J : MOMENT_INERTIA := 1.0E3;
        CF : DAMPING := 1.0E3 );
    PORT(
        TERMINAL T1, T2 : ELECTRICAL;
        TERMINAL M : ROTATIONAL_V;
        QUANTITY N : OUT REAL );
END ENTITY dc_motor;

ARCHITECTURE behave OF motor IS
    CONSTANT O2N : REAL := 60.0/(2.0*MATH_PI);
    QUANTITY V ACROSS IA THROUGH T1 TO T2;
    QUANTITY OMEGA ACROSS TSHAFT THROUGH M TO ROTATIONAL_V_REF;
BEGIN
    V == IA*RA + LA*IA'DOT + KE*OMEGA;
    TSHAFT == KE*IA - J*OMEGA'DOT - CF*OMEGA;
    N == O2N*OMEGA;
END ARCHITECTURE behave ;

```

Figure III.51: Modèle VHDL-AMS d'une machine à courant continu.

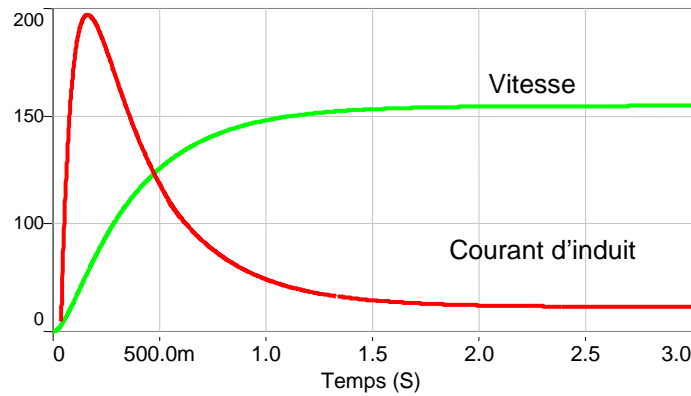


Figure III.52: Résultats de simulation d'une machine à courant continu par Simplorer.

On observe bien la pointe d'intensité au démarrage, la montée progressive en vitesse et, en régime établi, une vitesse de 162 rd/s pour un courant d'induit de 0,8A.

Le modèle VHDL-AMS pour une machine à courant continu donne des résultats identiques à ceux obtenus par macromodélisation.

III.9 Conclusion

Dans ce chapitre, nous avons décrit le cheminement qui nous a conduits à élaborer un outil spécifique à base de VHDL-AMS et de SPICE.

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

A travers ces travaux de simulation nous avons mis en évidence les propriétés des modélisations de SPICE ou de VHDL-AMS, et comparé leurs résultats qui montrent que les simulations obtenus pour les deux outils de modélisation sont quasi-identiques.

Chapitre IV

Boucle à verrouillage de phase (PLL)



IV.1 Introduction

Les boucles à verrouillage de phase (*Phase Locked Loop PLL*) sont des circuits intégrés très utilisés en électronique. Il s'agit donc comme leur nom l'indique d'un asservissement de phase dont le rôle est d'asservir la phase d'un oscillateur local à celle d'un signal extérieur. On trouve une boucle à verrouillage de phase dans tous les équipements modernes : récepteurs FM, décodeurs TV numériques, modems téléphoniques, ...

La Figure IV.1 représente le schéma fonctionnel d'une PLL de base. Il s'agit d'un système asservi à retour unitaire. Il existe plusieurs variantes de ce schéma en fonction des applications [2].

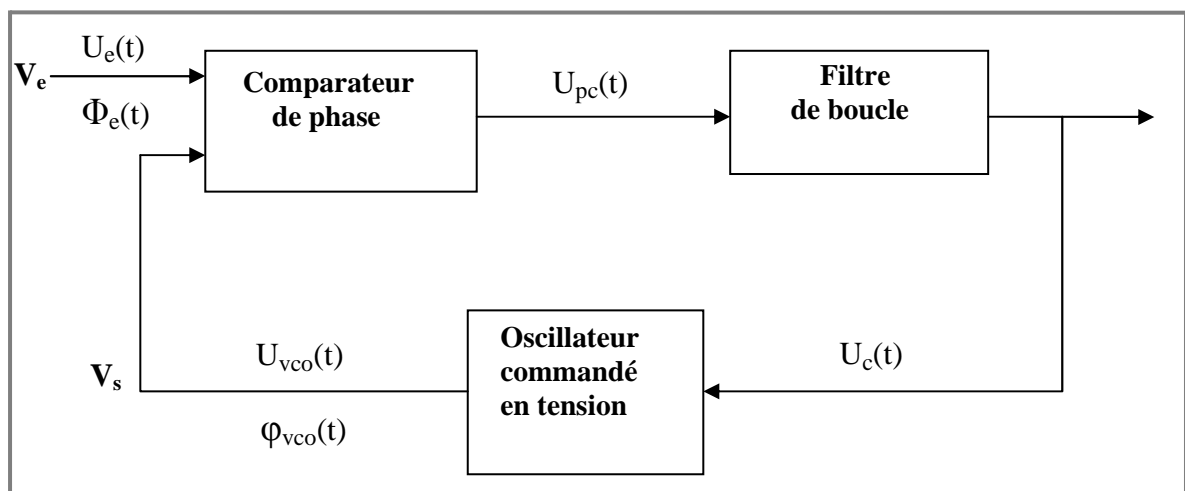


Figure IV.1: Schéma fonctionnel d'une PLL de base [2].

Les composants essentiels d'une PLL sont:

- ✚ Le détecteur ou comparateur de phase.
- ✚ Le filtre de boucle.
- ✚ L'oscillateur commandé en tension ou en courant (VCO).

IV.2 Détection d'évènements analogiques

Pour décrire un comparateur ou un convertisseur A/D, il est nécessaire de détecter avec précision le franchissement des seuils de quantification par le signal analogique d'entrée. La Figure IV.2 présente le comportement d'un comparateur à hystérésis de seuils V_1 et V_2 et de niveaux hauts et bas: V_{dd} et V_{ss} .

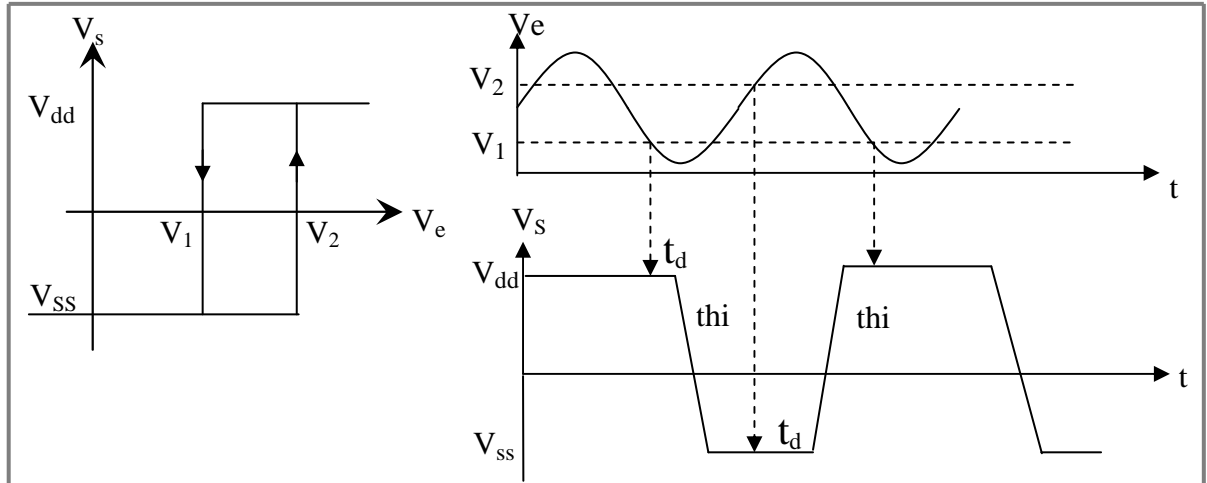


Figure IV.2: Modèle idéal du comparateur à hystérésis [4].

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

```
IF  $V_e < V_2$  THEN  $V_s = V_{ss}$  ELSE  $V_s = V_{dd}$ 
END IF
```

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

```
IF  $V_s(t-\Delta t) = V_{dd}$  AND ( $V_e(t-\Delta t) > V_1$  and  $V_e(t) < V_1$ ) THEN
     $V_s(t) = V_{ss}$ 
ELSE
    IF  $V_s(t-\Delta t) = V_{ss}$  AND ( $V_e(t-\Delta t) < V_2$  and  $V_e(t) > V_2$ ) THEN
         $V_s(t) = V_{dd}$ 
    END IF
END IF
```

Δt désigne l'intervalle de temps entre l'instant présent et le point précédent et $V(t - \Delta t)$ la valeur rendue par l'opérateur délai de paramètre Δt appliqué à la variable analogique V .

L'expression booléenne ($V_e(t - \Delta t) > V_1$ et $V_e(t) < V_1$) permet ainsi de détecter le franchissement descendant par V_e du seuil V_1 .

IV.3 Principe de fonctionnement

Le principe du verrouillage de phase est de forcer le signal de sortie de l'oscillateur commandé en tension (ou en courant) $u_{vco}(t)$ à suivre le signal d'entrée $u_e(t)$. Ceci signifie que les deux signaux ont mêmes fréquences et une différence de phases constante. Lorsque la PLL n'est pas verrouillée, les deux signaux peuvent avoir des fréquences différentes.

Le détecteur de phase est réalisé d'une manière simple. Les valeurs des deux formes d'onde d'entrée sont multipliées par le gain du détecteur de phase [10].

$$V_{out}(t) = gain \cdot vin1(t) \cdot vin2(t) \quad (IV.1)$$

$$u_{pc}(t) = K_d(\varphi_e - \varphi_s) = K_d \Delta\varphi \quad (IV.2)$$

Le comparateur de phase fournit à sa sortie une tension u alternative dont la valeur moyenne V donnée par un passe-bas est proportionnelle au déphasage entre V_e et V_s . Il est caractérisé par un coefficient souvent noté K_c défini par [10] :

$$K_c = \frac{\text{valeur moyenne de la tension en sortie}}{\text{déphasage entre les signaux d'entrée}} \quad \text{volts/radian} \quad (IV.3)$$

Ce signal est ensuite filtré pour générer une tension de commande, ou d'erreur, $u_c(t)$. Le filtre est un passe-bas d'ordre 1 (ceci fait que la PLL est un système asservi d'ordre 2), passif ou actif, dont le but est d'extraire la valeur moyenne du signal $u_{pc}(t)$ et ainsi d'obtenir un signal de commande $u_c(t)$ propre.

L'oscillateur commandé en tension est défini par une caractéristique $f_s(V_c)$ linéaire (du moins dans son domaine de fonctionnement) (Figure IV.3):

$$f_s - f_0 = K_{vco} V_c \quad (IV.4)$$

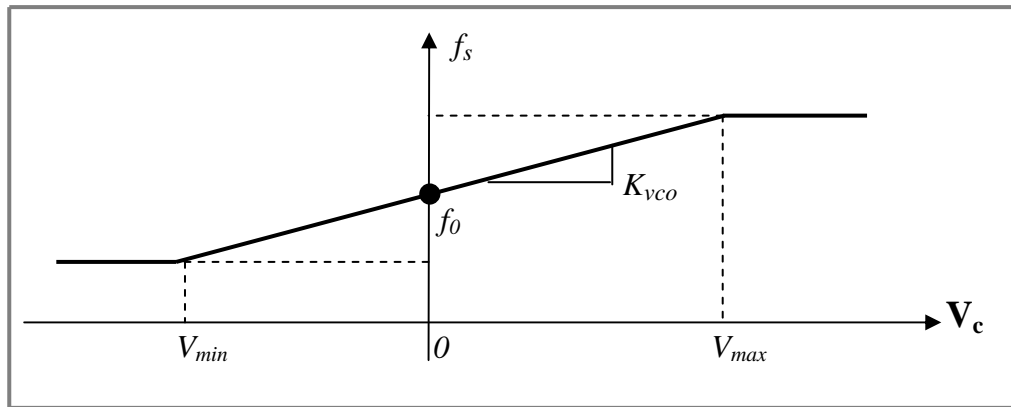


Figure IV.3: Caractéristique d'un oscillateur commandé en tension [2].

L'oscillateur VCO donne une fréquence qui varie en fonction de la tension de commande u_c appliquée sur son entrée. Il est linéarisé autour de f_0 et caractérisé par sa pente (*sensibilité*) K_0

$$K_{vco} = \frac{\text{variation de la pulsation du signal de sortie}}{\text{variation de la tension de commande}} \quad \text{Rad/(S.V)} \quad (IV.5)$$

f_0 est la **fréquence centrale** (ou la fréquence libre du VCO). La fréquence centrale doit être beaucoup plus grande que la fréquence de coupure du filtre de boucle.

La sensibilité est en général faible pour éviter des problèmes d'instabilité de la boucle. La fréquence étant la dérivée de la phase, on a aussi:

$$\frac{d\phi_{vco}}{dt} = K_{vco} u_{com} \quad (IV.6)$$

La sortie utile d'un tel système peut être soit le signal de commande de l'oscillateur (par exemple pour des fonctions de démodulation), soit le signal de sortie de l'oscillateur (par exemple pour des fonctions de génération de signaux stables).

Le fonctionnement d'une PLL comporte principalement deux modes: le **mode d'acquisition** ou de **capture** (*acquisition/capture mode*) et le **mode verrouillé** (*tracking mode*). En mode d'acquisition, la PLL cherche à synchroniser le signal généré par l'oscillateur avec celui fourni par l'entrée de référence. Une fois la synchronisation obtenue, la PLL entre dans le mode verrouillé pour lequel le signal généré par l'oscillateur suit tout changement de fréquence ou de phase du signal de référence. La Figure IV.4 illustre le fonctionnement d'une PLL en fonction de la caractéristique de l'oscillateur [2].

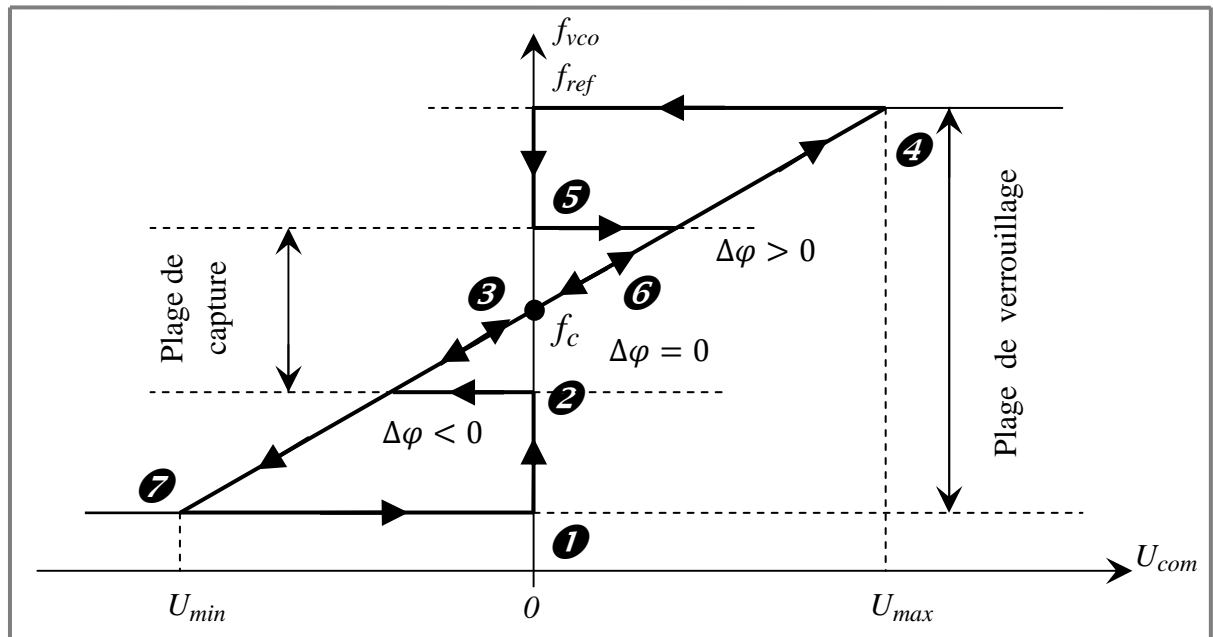


Figure IV.4: Modes de fonctionnement d'une PLL [2].

Tout d'abord, lorsque la tension d'entrée u_e est nulle ou constante, l'oscillateur oscille à sa fréquence centrale f_c . En effet, le déphasage $\Delta\phi$ vaut $\omega_{vco}t$. Cette composante est éliminée par le filtre de boucle et donc $u_c = 0$.

Faisons maintenant croître la tension d'entrée à partir de $u_e = 0$ (point 1 sur la Figure IV.4) en supposant un régime de fonctionnement sinusoïdal et un comparateur de phase réalisé sous la forme d'un multiplieur analogique. Le signal $u_{pc}(t)$ comporte ainsi une composante à la fréquence $f_e + f_{vco}$ et une autre à la fréquence $f_e - f_{vco}$. Jusqu'au point 2, le filtre passe-bas, supposé idéal, élimine ces deux composantes.

Au point 2, la composante basse fréquence arrive dans la bande passante du filtre. Le signal de commande devient une composante variable à cette fréquence. Ceci a pour effet de baisser la fréquence du signal généré par l'oscillateur jusqu'au point où la fréquence de battement $f_e - f_{vco}$ devient nulle. La PLL est alors verrouillée. En réalité la baisse de la fréquence de l'oscillateur ne se fait pas immédiatement (filtre non idéal) et la fréquence de début du processus de verrouillage est appelée la **fréquence de capture** ou **d'acquisition**.

Au point 3, la PLL est verrouillée et toute augmentation de la fréquence d'entrée se traduit par une augmentation identique de la fréquence de l'oscillateur.

Le déverrouillage de la PLL a lieu lorsque l'on atteint la non linéarité des éléments (point 4). Dès ce moment, l'oscillateur se remet à osciller à sa fréquence centrale, car la composante basse fréquence à la sortie du comparateur de phase est à nouveau filtrée.

Un raisonnement identique permet de déterminer le comportement de la PLL pour une tension d'entrée alternative à fréquence décroissante (points 5, 6 et 7). Il faut remarquer que la caractéristique non idéale du filtre de boucle a pour effet de provoquer des oscillations croissantes de la tension u_c avant le verrouillage et après le déverrouillage.

Chacun des composants de la PLL peut être réalisé de manière analogique ou logique. Selon les choix on parle de PLL analogique, logique ou hybride (mixte) [2].

IV.4 Modélisation d'une PLL

La PLL sera donc modélisée à partir de ses fonctions de base : le multiplieur, le VCO et le filtre. Le multiplieur ou détecteur de phase est décrit par la multiplication de deux signaux d'entrée, l'un représentant le signal à démoduler, l'autre étant le signal sortant du VCO. La sortie de cette fonction est « échantillonnée » entre -1 et $+1$ V. Le multiplieur est caractérisé par son facteur de conversion phase – tension K_c .

Le VCO est décrit par une fonction sinusoïdale dont le terme de déphasage est contrôlé par la tension de commande. Le VCO est caractérisé par sa fréquence libre F_0 et son facteur de conversion tension-fréquence K_{vco} .

Le filtre du premier ordre est quant à lui décrit par sa Transformée de Laplace. Ce filtre, de type RC, est défini par sa constante de temps τ [1].

IV.4.1 Modèle VHDL-AMS de la PLL

IV.4.1.1 Comparateur de phase

La forme la plus courante d'un comparateur de phase analogique est un multiplieur à quatre quadrants. Soit les deux signaux d'entrée tels que [2]:

$$u_e = U_e \sin(\omega_e t + \varphi_e) \quad (IV.7)$$

$$u_{vco} = U_{vco} \sin(\omega_e t + \varphi_{vco}) \quad (IV.8)$$

La sortie du multiplieur est alors:

$$u_{pc} = K_c U_e U_{vco} = \frac{K_c U_e U_{vco}}{2} [\cos(\varphi_e - \varphi_{vco}) + \cos(2\omega_e t + \varphi_e + \varphi_{vco})] \quad (IV.9)$$

Le second terme est éliminé par le filtre de boucle, ce qui donne [2]:

$$u_{pc}(t) = \frac{K_c U_e U_{vco}}{2} \cos(\varphi_e - \varphi_{vco}) = K_m \cos(\Delta\varphi) \quad (IV.10)$$

Pour des petites valeurs de $\Delta\varphi$ la tension de sortie est proportionnelle à l'erreur de phase.

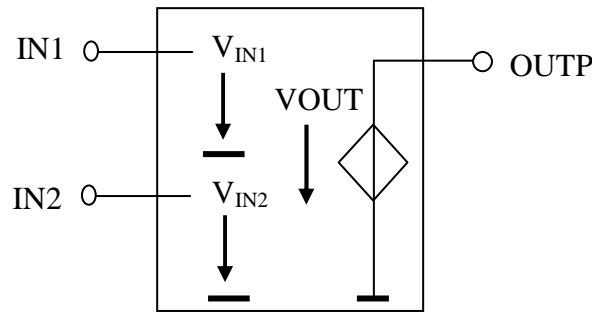


Figure IV.5: Structure du modèle de base du détecteur de phase [10].

La figure IV.6 donne le modèle VHDL-AMS du comparateur de phase analogique.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY phase_detect IS
  GENERIC (GAIN : REAL := 1.0);
  PORT (TERMINAL IN1 : ELECTRICAL;
        TERMINAL IN2 : ELECTRICAL;
        TERMINAL OUTP : ELECTRICAL);
END ENTITY phase_detect;

ARCHITECTURE basic OF phase_detect IS
  QUANTITY Vin1 ACROSS IN1;
  QUANTITY Vin2 ACROSS IN2;
  QUANTITY Vout ACROSS Iout THROUGH outp;
BEGIN
  VOUT == GAIN*VIN1*VIN2;
END ARCHITECTURE basic;

```

Figure IV.6: Modèle VHDL-AMS du détecteur de phase analogique.

IV.4.1.2 Filtre de boucle

Le filtre passe-bas a une influence importante sur le régime transitoire et détermine en grande partie les performances de l'asservissement. Les fonctions de transfert couramment utilisées sont données par [2]:

$$\text{Type 1: } H(p) = H_0 \cdot \frac{1}{1+j\omega\tau_1} \quad (\text{IV.11})$$

$$\text{Type 2: } H(p) = H_0 \cdot \frac{1+j\omega\tau_2}{1+j\omega(\tau_1+\tau_2)} \quad (\text{IV.12})$$

$$\text{Type 3: } H(p) = H_0 \cdot \frac{1+j\omega\tau_2}{j\omega\tau_1} \quad (\text{IV.13})$$

$$\text{Type 4: } H(p) = H_0 \cdot \frac{1}{j\omega\tau_1} \quad (\text{IV.14})$$

Avec $\tau_1 = R_1.C$ et $\tau_2 = R_2.C$. Il serait possible d'écrire un modèle plus général permettant de réaliser les quatre types de filtres.

```

LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;

ENTITY filtre IS
  GENERIC(GAIN : REAL := 1.0;           -- gain
          FC : REAL := 10.0e3;         -- fréquence centrale [HZ]
          ZF : REAL := 1.0);
  PORT(TERMINAL inp : ELECTRICAL;
        TERMINAL outp : ELECTRICAL);
END ENTITY filtre;

ARCHITECTURE behav OF filter IS
  -- description d'un filter
  QUANTITY VIN ACROSS inp;           -- ouverture du sortir branché
  QUANTITY Vout ACROSS Iout THROUGH outp; -- sortie branché
  BEGIN
    Vout == VIN*LTF((0 => GAIN), (0 => 1.0, 1 => 1.0/MATH_2_PI/FC));
  END ARCHITECTURE behav;

```

Figure IV.7: Modèle VHDL-AMS du filtre de boucle.

IV.4.1.3 Oscillateur commandé en tension

Un oscillateur analogique commandé en tension (VCO) génère un signal sinusoïdal dont la fréquence est proportionnelle à la valeur d'une tension de commande. La caractéristique linéaire d'un VCO s'exprime par la relation [2]:

$$\omega_{vco}(t) = \omega_c + K_{vco}(u_c(t) - U_{c0}) \quad (\text{IV.15})$$

Où ω_c est la pulsation centrale et U_{c0} la valeur de la tension de commande permettant l'oscillation à la fréquence centrale. Le calcul de la phase est alors fait par :

$$\varphi_{vco}(t) = \int \omega_{vco}(t) dt = \omega_c t + K_{vco} \int (u_c(t) - U_{c0}) dt \quad (\text{IV.16})$$

Le signal de sortie de l'oscillateur est ainsi donné par [2]:

$$u_{vco}(t) = U_{vco} \sin(\varphi_{vco}) \quad (\text{IV.17})$$

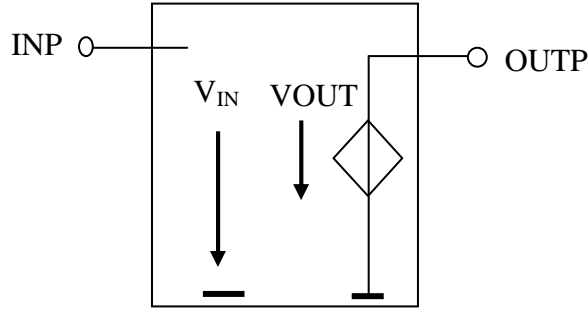


Figure IV.8: Structure du modèle de base du VCO [10].

La figure IV.9 donne le modèle VHDL-AMS du VCO réalisant les équations ci-dessus.

```

LIBRARY IEEE;
USE IEEE.MATH_REAL.ALL;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY vco IS
    GENERIC (GAIN : REAL := 1.0;           -- gain du vco [-]
             KVCO: REAL := 10.0e6;        -- pente (sensibilité) du vco [HZ/V]
             FC : REAL := 1.0E3;          -- fréquence centrale [HZ]
             UC0 : REAL := 1.2            -- tension de commande a F=FC [V]
             );
    PORT (TERMINAL INP, OUTP: ELECTRICAL);
END ENTITY vco;

ARCHITECTURE behav OF VCO IS
    CONSTANT WC : REAL := MATH_2_PI * FC;
    CONSTANT KVC : REAL := MATH_2_PI * KVCO;           -- [RAD/(S*V)]
    QUANTITY Uin ACROSS INP TO GROUND;
    QUANTITY OUTP ACROSS Iout THROUGH OUTP TO GROUND;
    QUANTITY DP, P: REAL;                               -- PHASE
BEGIN
    DP == WC + KVC *(UIN - UC0);
    P == DP'INTEG;
    UOUT == GAIN * SIN(P);
END ARCHITECTURE behav;

```

Figure IV.9: Modèle VHDL-AMS du VCO analogique.

IV.4.2 Modèle Pspice de la PLL

La figure IV.10 représente la modélisation comportementale de la PLL. On voit apparaître les trois modules élémentaires : détecteur de phase, VCO et filtre avec leurs paramètres respectifs.

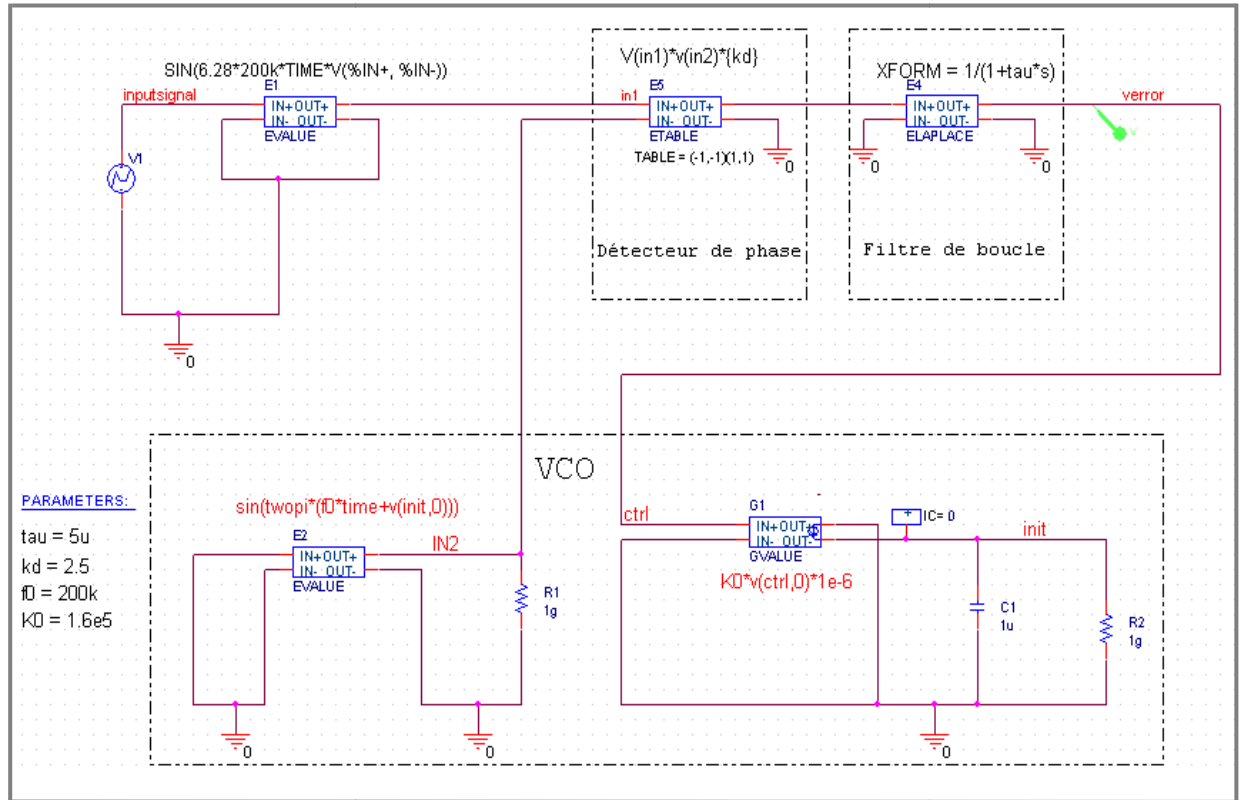
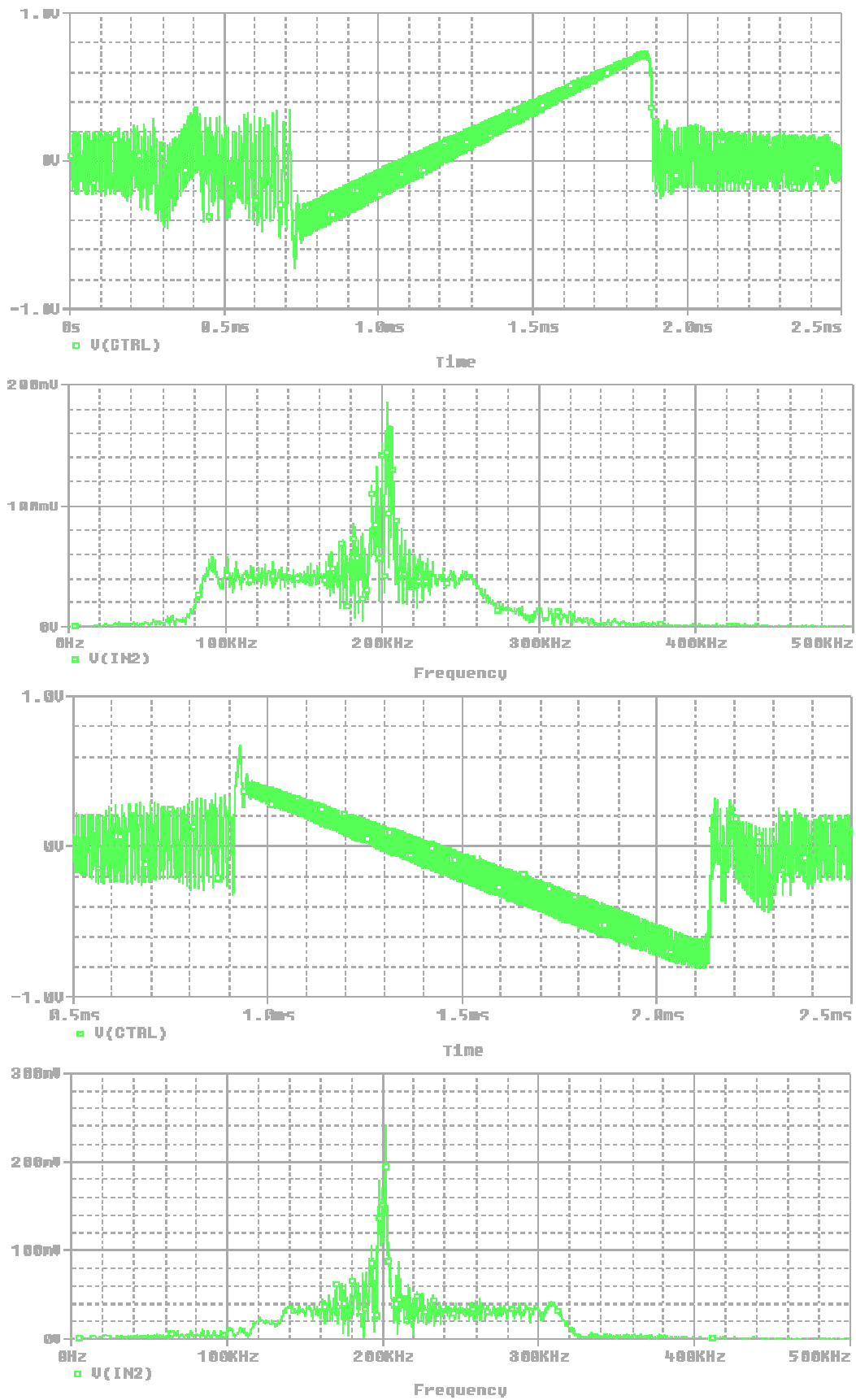


Figure IV.10: Modèle Pspice de la PLL.

Concernant les simulations à effectuer, on s'attachera tout d'abord à mettre en évidence les plages de verrouillage et de capture. Il faut donc faire varier la fréquence de la source de tension à l'entrée de la PLL. La PLL n'accroche pas tant que la fréquence du signal d'entrée n'a pas atteint la fréquence $f1$ de capture et le VCO oscille alors à sa fréquence libre $f0$. A partir de cette fréquence $f1$, la PLL est verrouillée et la fréquence du VCO suit la fréquence d'entrée du signal (signaux déphasés). Ce déphasage se traduit par l'apparition d'une valeur moyenne à la sortie du filtre (Figure IV.11(a)). A partir de $f3$, la PLL se « déverrouille » et le VCO revient à sa fréquence libre $f0$. En faisant varier les fréquences d'entrée par valeur décroissante, on obtient de la même manière $f2$ et $f4$. Ces variations de fréquence du VCO se visualisent aisément grâce à la FFT du signal de sortie du VCO (Figure IV.11(b)).



(a)

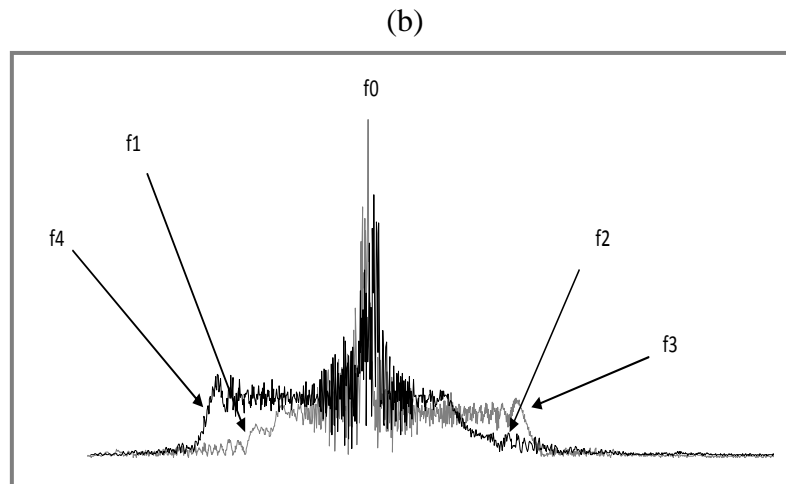


Figure IV.11 : (a) Tension de sortie du filtre (fréquences d'entrée croissantes et décroissantes) et (b) Réponse en fréquence du VCO.

L'écriture d'un programme VHDL-AMS pour le PLL donne des simulations identiques à ceux obtenues par macromodélisation.

IV.5 Conclusion

Ce chapitre présente deux approches concernant la modélisation comportementale d'un circuit, en l'occurrence la PLL. Bien que les outils soient très différents, la méthode de modélisation est tout à fait identique : décomposition de la structure en trois fonctions (détecteur de phase, VCO et filtre). Si le modèle développé sous PSPICE semble facile à mettre en œuvre, il est évident que l'approche VHDL-AMS permet d'obtenir un modèle transportable d'un simulateur à l'autre.

Conclusion Générale



Conclusion générale

Devant la réduction constante des temps de gestation des projets industriels, la miniaturisation et la complication galopante qui l'accompagne, les outils de CAO multi-domaines, multi-signaux et multi-abstractions sont devenus une nécessité pour améliorer la productivité et réduire les coûts de développement à l'aide du prototypage virtuel.

Les outils de CAO analogiques doivent évoluer pour rattraper leur retard sur la conception numérique qui est de nos jours largement automatisée. Cette automatisation est devenue possible grâce au développement des langages de description matériel standard tel que le VHDL associé aux outils de synthèse numérique, et surtout à la nature même des circuits numériques pour lesquels les niveaux inférieurs sont relativement figés.

L'extension de ces standards pour les circuits analogiques et mixtes (norme VHDL-AMS) ouvre la voie à l'amélioration des outils de conception analogique et mixte et peut participer au développement des outils de synthèses analogiques et mixtes.

Le langage VHDL-AMS, encore jeune, et les outils les plus récents qui lui sont associés permettent aujourd'hui d'envisager la construction, par les spécialistes du langage, de bibliothèques de modèles généralistes dans lesquelles les concepteurs de systèmes pourront puiser, sans pour autant avoir besoin d'une connaissance approfondie des HDLs sous jacents.

La simulation et la modélisation sont l'un des domaines clé déterminant le succès de la conception des CI analogiques.

Nous avons vu les principes de la modélisation comportementale ainsi que son intérêt. Les modèles comportementaux de haut niveau permettent de décrire avec une précision suffisante les phénomènes physiques essentiels pour une application donnée. La description structurelle bas niveau (de type SPICE) serait pénalisante en termes non seulement de temps de simulation, mais aussi de confidentialité.

Nous avons montré que le langage VHDL-AMS est particulièrement bien adapté à ce niveau de modélisation ; il permet en outre la réutilisation de modèles numériques VHDL sans modification.

Nous avons présenté deux techniques de modélisation : La macromodélisation comportementale PSPICE-ORCAD et l'utilisation du langage VHDL-AMS. Nous avons montré que malgré que la méthode de modélisation est identique quels que soient les outils, et que les modèles développés sous SPICE au avec le VHDL-AMS sont relativement facile à concevoir les résultats de simulation sont quasiment identique. Cependant, le VHDL-AMS permet d'élaborer des modèles transportable d'un simulateur à l'autre ce qui offre aux concepteurs une grande liberté de conception.

ABM	<i>Analog Behavioral Modeling</i>
AMS	<i>Analog and Mixed-Signal</i>
AC	<i>Alternative Current</i>
AO	<i>Amplificateur Oopérationnel</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
CAD	<i>Computer-Aided-Design</i>
CAO	<i>Conception Assistée par Ordinateur</i>
DAE	<i>Differential and Algebraic Equations</i>
DC	<i>Direct Current</i>
DLL	<i>Dynamic Link Library</i>
FPGA	<i>Field-Programmable Gate Array</i>
GUI	<i>Graphical User Interface</i>
HAMSter	<i>High performance AMS tool for Engineering and Research</i>
HDL	<i>Hardware Description Language</i>
IC	<i>Integrated Circuit</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
MRT	<i>Minimum Resolvable Time</i>
PV	<i>Prototypage Virtuel</i>
RF	<i>Radio Frequency</i>
SOC	<i>System On Chip</i>
SPICE	<i>Simulation Program with Integrated Circuit Emphasis</i>
PLL	<i>Phase Locked Loop</i>
VCO	<i>Voltage Controlled Oscillator</i>
VHDL	<i>VHSIC HDL</i>
VHSIC	<i>Very-High-Speed Integrated Circuit</i>
VHDL-AMS	<i>VHSIC HDL Analog and Mixed-Signal</i>

Bibliographie



Bibliographie

- [1] D. Andreu, ***"La Modélisation Comportementale de SPICE a VHDL-AMS"***, ENSEEIHT, Département d'électronique, Toulouse.
- [2] Alain Vachoux, ***"Modélisation de Systèmes Intégrés Analogiques et Mixtes"***, Laboratoire de Systèmes Microélectroniques EPFL 2002.
- [3] Samia Belkacem, ***"Macromodélisation Comportementale De Circuits Analogiques Application Au Circuit Convoyeur De Courant"***, Magister en micro-électronique université de Batna, 2005.
- [4] F. Lémery, ***"Modélisation Comportementale des Circuits Analogiques et Mixtes"***, Thèse de doctorat, Institut National Polytechnique De Grenoble, Décembre 1995.
- [5] S. Jemmali, ***" Contribution A L'élaboration De Méthodologies Et D'outils D'aide A La Conception De Système Multi Technologiques"***, Thèse de doctorat, Ecole Nationale Supérieure Des Communications, 2003.
- [6] S. Snaidero, ***"Modélisation multidisciplinaire VHDL-AMS de systèmes complexes : vers le Prototypage Virtuel"***, thèse doctorat de l'université de louis pasteur Strasbourg I.
- [7] ***" ORCAD PSPICE SIMULATION ANALOGIQUE / LOGIQUE"***, Documentation ORCAD, fichier Manual.
- [8] Jacques Rouillard, ***"LIRE & comprendre VHDL-AMS"***, 2008.
- [9] O. Alali, ***"Modélisation VHDL-AMS analogique et simulation SPICE"***, Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, 1998.
- [10] R. Frevert , ***" MODELING AND SIMULATION FOR RF SYSTEM DESIGN"***, Springer, 2005.
- [11] Ansoft Corporation SIMPLORER présentation [http : // ww.ansoft.com/products/](http://www.ansoft.com/products/).
- [12] David A. Johns, Ken Martin, ***"Analog Integrated Circuit Design"***, John Willy & Sons Inc, 2000.
- [13] ***"Simulation System Simplorer 7.0 User Manual"***, Documentation Simplorer, fichier Manual.
- [14] ***" Simplorer 7.0 VHDL-AMS Tutorial "***, Documentation Simplorer.
- [15] ***"SIMPLORER C-interface"***, Fichiers d'aide de Simplorer, fichier C_interf.hlp.

- [16] Paul Tobin, "*PSpice for Circuit Theory and Electronic Devices*", Morgan Claypool Publishers (Jan 2007).
- [17] Paul Tobin, "*PSpice for Digital Communications Engineering*", Morgan Claypool Publishers (Feb 2007).
- [18] Paul Tobin, "*PSpice for Analog Communications Engineering*", Morgan Claypool Publishers (March 2007).
- [19] Céline Guillemot, "*Etude et intégration numérique d'un système multicapteurs Amrc de télécommunication base sur un prototype virtuel Utilisant le langage de haut niveau VHDL-AMS*", thèse doctorat de l'université de Toulouse p.
- [20] V.F. Kroupa, "*Phase Lock Loops and Frequency Synthesis*", Academy of Sciences of the Czech Republic, Prague.
- [21] Richard Perdriau, "*Méthodologie de prédiction des niveaux d'émission conduite dans les circuits intégrés, à l'aide de VHDL-AMS*", thèse doctorat de l'université Catholique de Louvain p.
- [22] R. Frevert, "*Modeling And Simulation For Rf System Design*", Springer 2005.
- [23] S. Snaidero, "*Modélisation multidisciplinaire VHDL-AMS de systèmes complexes : vers le Prototypage Virtuel*", thèse doctorat de l'université de Louis Pasteur Strasbourg I.
- [24] M. Rashid "*SPICE for Power Electronics and Electric Power*" Second Edition.
- [25] P J Randewijk and H du T Mouton "*USING VHDL-AMS FOR ELECTRICAL, ELECTROMECHANICAL, POWER ELECTRONIC AND DSP-ALGORITHM SIMULATIONS*", University of Stellenbosch, Dept. of Electrical and Electronic Engineering, Stellenbosch, South Africa.

Résumé

La conception de système passe par la description comportementale des différentes parties (sous-systèmes) du système et de leurs intercommunications. Le langage VHDL-AMS, dont la norme est en train de sortir, vient à point pour favoriser cette description et ces échanges.

Ce mémoire a pour objectif :

1. De présenter le passage de la simulation analogique classique à la simulation comportementale et de savoir comment transformer un simulateur électrique de type SPICE en simulateur comportemental.

2. La modélisation des composants et des blocs par SPICE et le langage VHDL-AMS.

Il répond au désir de pouvoir fournir dès à présent, alors que la norme VHDL-AMS est sur le point de sortir officiellement, un outil de simulation et une ébauche de méthode de modélisation VHDL-AMS.

Le mémoire comprend donc deux grandes parties. Dans la première partie intitulée « Simulation Analogique et Comportementale », nous avons étudié le fonctionnement et la structure d'un simulateur électrique analogique (SPICE), et les grands principes (analogiques) du langage VHDL-AMS.

Une comparaison entre les résultats de simulation pour les deux applications soit par SPICE ou VHDL-AMS a montrée que les résultats sont les même quelque soit l'outil de simulation.

Mots clés : VHDL-AMS, SPICE, Modélisation, simulation.

Abstract

The design of systems is described by the behavioral of the various parts (subsystems) and their interconnections. The language VHDL-AMS, whose standard is leaving, comes at the right moment to support this description and these exchanges.

This memory aims at:

1. To present the passage of traditional analogical simulation to behavioral simulation and of knowing how to transform an electric simulator type SPICE into behavioral simulator.

2. The modeling of the components and the blocks by SPICE and VHDL-AMS language.

It answers the desire to be able to provide as of now, whereas standard VHDL-AMS are about to leave officially, a tool for simulation and an outline of method of modeling VHDL-AMS.

The memory thus comprises two main parts. In the first part, we studied the operation and the structure of an analogical electric simulator (SPICE), and the great principles (analogical) of language VHDL-AMS.

A comparison between the results of simulation for the two applications is by SPICE and VHDL-AMS showed that the results are the same for the two tools of the simulation. The transportable operation of the VHDL-AMS gives it the first class.

Keywords: VHDL-AMS, SPICE, Modeling, simulation.