CONTRIBUTION A L'ELABORATION DE METHODOLOGIES ET D'OUTILS D'AIDE A LA CONCEPTION DE SYSTEMES MULTI-TECHNOLOGIQUES

Sabeur Jemmali

Thèse

présentée pour obtenir le grade de docteur de l'Ecole nationale supérieure des télécommunications

Spécialité : Electronique et Communications

Sabeur Jemmali

Contribution à l'élaboration de méthodologies et d'outils d'aide à la conception de systèmes multi-technologiques

Soutenance le 27 novembre 2003 devant le jury composé de

Yves Danto	Président
Carles Ferrer Nouri Masmoudi	Rapporteurs
Michèle Trân Andrei Vladimirescu Jean Oudinot	Examinateurs
Jean-Jacques Charlot	Directeur de thèse

Ecole Nationale Supérieure des Télécommunications

A mes parents,
à mes sœurs,
à mon frère,
et à mon épouse Houda

REMERCIEMENTS

Le travail présenté dans cette thèse a été effectué au sein de l'équipe Modélisation Comportementale pour le Design (MCD) du Département Communications et Electronique (COMELEC) de l'Ecole Nationale Supérieure des Télécommunications (TELECOM PARIS) et dans le cadre du Projet européen SPARTE.

Je remercie Monsieur Yves DANTO, Professeur des Universités à l'Université Bordeaux 1, responsable de l'opération fiabilité de l'IXL, UMR 5818, Laboratoire d'Intégration des Composants et Systèmes, pour l'intérêt qu'il a porté à mon travail et pour l'honneur qu'il m'a fait en présidant le jury de cette thèse. Qu'il trouve ici l'expression de ma profonde gratitude.

Tous mes remerciements vont à Messieurs Carles FERRER et Nouri MASMOUDI leur lecture attentive du manuscrit en tant que rapporteurs de cette thèse.

Je tiens à remercier Madame Michèle TRÂN, Ingénieur à MBD.A qui a accepté d'être rapporteur de cette thèse. Je le remercie profondément pour ses remarques, ses conseils, ses critiques pertinentes et constructives, et de m'avoir aider à corrigé ce mémoire.

J'exprime mes profonds remerciements à Monsieur Andrei VLADIMIRESCU, Professeur au *Department of Electrical Engineering and Computer Sciences* à l'Université de California à Berkeley, d'avoir accepté d'examiner ce travail et pour les échanges enrichissants que j'ai pu avoir avec lui.

Mes remerciements s'adressent également à Monsieur Jean OUDINOT, Docteur en électronique et responsable marketing européen à Mentor Graphics, pour avoir accepté d'être l'un des examinateurs de ce travail.

Mes plus vifs remerciements ainsi que ma profonde reconnaissance vont au Professeur Jean-Jacques CHARLOT, pour m'avoir initié à la recherche et j'espère que ces quelques années passées sous sa direction m'ont permis d'acquérir un peu de son expérience scientifique et humaine. Il m'a beaucoup aidé dans la rédaction des publications et de ce mémoire. Je n'oublie pas les nombreuses discussions sur ces voyages à travers le monde.

Mes remerciements s'adressent au personnel administratif du département et au chef du département Jean-Claude BIC, en espérant un bon établissement. Mes vifs remerciement et tout particulièrement à Chantal CADIAT, Danielle CHILDZ et Marie BAQUERO.

Mes remerciements s'adressent aussi à tous les membres du département et plus particulièrement Jacky PORTE, Jean-François NAVINER, Lirida NAVIER, Patricia DESGREYS, Hervé PETIT, Patrick LOUMEAU, et Jean PROVOST. Egalement, je remercie Ali NEHME pour les échanges scientifiques et d'ordre général. Sans oublier de remercier l'inséparable Karim BEN KALAIA (Kilo), le véritable ami et le compagnon de métro, de m'avoir me supporté tout au long de ma thèse.

Enfin, il m'est très difficile de remercier à sa juste valeur toute ma famille de thésards et de stagiaires avec qui j'ai pu passer les meilleurs moments de travail, de joie, et de bonheur. Je renouvelle mes remerciements à Mohamed, Nesrine, Elisabeth, Frédéric, Kae, Van Tam, Belgesim, Sompasong, Louis, et Souhaila. Je remercie infiniment mes amis Walid et Bilal de m'avoir initié à la recherche et de les avoir trouver à coté de moi dans les moments difficiles.

CONTRIBUTION A L'ELABORATION DE METHODOLOGIES ET D'OUTILS D'AIDE A LA CONCEPTION DE SYSTEMES MULTI-TECHNOLOGIQUES

RESUME

Cette thèse a pour thème la contribution à l'élaboration de méthodologies et d'outils d'aide à la conception de systèmes multi-technologiques. Ces travaux de recherche et de développement s'inscrivent dans le cadre d'un projet européen portant sur la simulation basée sur les spécifications et les indicateurs de performance au regard des effets thermiques et électriques. L'objectif est de créer une plate-forme de modélisation permettant de voir les modèles certifiés et de répondre aux critères fixés en se concentrant sur la fonction, le comportement et la structure, et la physique du composant. Cette plate-forme met en oeuvre des procédures (langage VHDL-AMS, méthodologies, …) et des ressources (outils de CAO, bibliothèques, …). Une telle plate-forme repose sur des bases conceptuelles alliant méthodes de conception (approches descendante et ascendante) et méthodes de modélisation (fonctionnelle, comportementale et structurelle, physique).

MOTS CLES : VHDL-AMS, SPICE, Méthodologie de conception, Multi-technologie, Colorimétrie.

CONTRIBUTION FOR BUILDING METHODOLOGIES AND DESIGN TOOLS IN MULTI-TECHNOLOGIQUES SYSTEMS

ABSTRACT

This thesis has for theme the contribution for building methodologies and tools for designing multi-technological systems. These research and development works take place in a European Project concerning based simulation on specifications and performance indicators versus thermal and electric effects. The objective is to create a modelling platform showing the certified models and to reply to the set up criteria in focussing on the function, the behaviour & structure, and the physics of the component. This platform implements procedures (language VHDL-AMS, methodologies,…) and resources (CAO tools, libraries, …). Such a platform is based on conceptual bases gathering design methods (top/down and bottom/up approaches) and modelling methods (at functional, behavioural and structural, and physical levels).

KEY WORDS : VHDL-AMS, SPICE, Design Methodology, Multi-technology, Colorimetry.

SOMMAIRE

REMERCIEMENTS **3**

RESUME **5**

INTRODUCTION GENERALE **10**

CONTEXTE GENERAL DE L'ETUDE 10
Contexte académique *10*
Contexte industriel européen *11*
OBJECTIF 11
STRUCTURE DU MEMOIRE DE THESE 11
Première partie : Méthodes & Outils *11*
Deuxième partie : Applications *12*

PREMIERE PARTIE : METHODES & OUTILS **13**

CHAPITRE 1 ..... **15**

1 METHODES ET LANGAGES : VERS L'UTILISATION DU LANGAGE VHDL-AMS **15**

1.1 INTRODUCTION : CONTEXTE DE L'ETUDE 15
1.2 MODELES ET SIMULATEURS 16
1.2.1 Modélisation et simulation *16*
1.2.2 Limitations de SPICE *17*
1.2.3 Les différents types de langages *17*
1.2.3.1 Les langages de description logicielle 17
1.2.3.2 Le langage de description structurelle (macro-modélisation) 18
1.2.3.3 Les langages de description matérielle 20
1.3 BREVE PRESENTATION DU LANGAGE VHDL-AMS 22
1.3.1 Introduction *22*
1.3.2 Modélisation comportementale analogique *22*
1.3.2.1 Les systèmes analogiques conservatifs 23
1.3.2.2 Les systèmes analogiques non-conservatifs 23
1.3.3 Structure d'un modèle VHDL-AMS *23*
1.3.4 Classes d'objets *25*
1.3.4.1 QUANTITYs 25
1.3.4.2 TERMINALs et NATUREs 25

1.3.5 Déclarations simultanées *26*
1.4 APPORTS DE VHDL-AMS 30
1.5 APPLICATION A LA MODELISATION HIERARCHIQUE 30
1.6 CONCLUSION 31

CHAPITRE 2 34

2 OUTIL : VERS LA CONSTRUCTION DE VAMSPICEDESIGNER 34

2.1 HISTORIQUE ET INTRODUCTION 34
2.1.1 Historique : BVHDLA *34*
2.1.2 Introduction *35*
2.2 COMPILATEUR VAMSPICE 35
2.2.1 Principe du compilateur *35*
2.2.1.1 Phases d'analyse 35
2.2.1.2 Production de code 36
2.2.1.3 Gestion de la table des symboles 36
2.2.1.4 Détection et compte rendu des erreurs 36
2.2.2 VamSpice (VamSpiceCompiler) *38*
2.2.2.1 Principe 38
2.2.2.2 Méthodologie 38
2.2.2.3 Création d'un modèle VHDL-AMS 40
2.3 VamSpiceEditor 48
2.4 LE SIMULATEUR SPICE OPUS (= SPICE3 + XSPICE + OPTIMISATION + *NUTMEG*) 50
2.5 LA SCHEMATIQUE « *ELECTRIC*^{TM} VLSI DESIGN SYSTEM » 50
2.5.1 Icône, VHDL-AMS, circuit, et netlist SPICE *50*
2.5.2 Macro-composant, icône et hiérarchie *53*
2.5.3 Une bibliothèque VHDL-AMS (VHDL-AMS Libraries Parts) *53*
2.5.3.1 Organisation de la bibliothèque 54
2.5.3.2 Transport du composant 55
2.6 LIEN ENTRE LES OUTILS 60
2.7 EXEMPLES D'APPLICATION 62
2.7.1 Modélisation fonctionnelle *62*
2.7.1.1 Valeur absolue 62
2.7.1.2 Tangente hyperbolique 62
2.7.1.3 Différenciateur-multiplieur 63
2.7.1.4 Comparateur de tension 64
2.7.1.5 Intégrateur-gain 64
2.7.2 Modélisation comportementale *65*
2.7.2.1 Résistance 65
2.7.2.2 Capacité 65
2.7.2.3 Filtre passif 1er ordre 66
2.7.2.4 Diode 67
2.7.2.5 Transistor MOS correspondant au niveau 1 de modélisation SPICE 68
2.7.2.6 Interrupteur (*switch model*) 68
2.7.2.7 Convertisseur Flash 69
2.7.3 Modélisation physique *71*
2.7.3.1 Résistance 71
2.8 CONCLUSION 71

DEUXIEME PARTIE : APPLICATIONS 73

CHAPITRE 3 74

3 MODELISATION – SIMULATION DE DISPOSITIFS MOS AVEC VHDL-AMS ET AVEC SPICE 74

3.1 TRANSISTOR MOS 74
3.1.1 Introduction 74
3.1.2 Modèle SPICE niveau 3 du transistor MOS 79
3.1.3 Implantation VHDL-AMS du modèle SPICE niveau 3 (voir annexe 1) 81
3.2 CELLULE MEMOIRE DE TYPE EEPROM 82
3.2.1 Introduction 82
3.2.2 Modélisation de la cellule EEPROM 83
3.2.2.1 Transistor de sélection 84
3.2.2.2 Transistor d'état 84
3.2.3 Implantation du modèle dans VHDL-AMS (voir annexe 2) 91
3.2.3.1 Régime statique 91
3.2.3.2 Régime transitoire 92
3.3 MODELE UNICELL DU TRANSISTOR MOS 97
3.4 CONCLUSION 99

CHAPITRE 4 102

4 MODELISATION – SIMULATION ELECTRO-THERMIQUE DES MEMOIRES FLASH ET SDRAM 102

4.1 INTRODUCTION 102
4.2 METHODOLOGIE GENERALE DE LA SIMULATION ELECTRO-THERMIQUE 102
4.2.1 Introduction 102
4.2.2 Analyse de la méthode explicite brute 103
4.2.3 Variante de la méthode explicite : méthode en deux étapes 104
4.3 APPLICATION DE LA METHODE EXPLICITE A DEUX ETAPES A UNE MEMOIRE FLASH 106
4.3.1 Modélisation d'une cellule de mémoire FLASH 106
4.3.2 Analyse avec ADVanceMS de Mentor Graphics 107
4.4 MODELE COMPORTEMENTAL DE CALCUL ELECTRO-THERMIQUE DES MEMOIRES FLASH ET SDRAM A L'AIDE DE LA METHODE EXPLICITES A DEUX ETAPES 109
4.5 APPLICATION AU TRAITEMENT ELECTRO-THERMIQUE D'UNE MEMOIRE SDRAM A L'AIDE DE LA METHODE EXPLICITE A DEUX ETAPES AVEC VAMSPICEDESIGNER 110
4.5.1 Circuit équivalent de la mémoire SDRAM pour calculer les puissances et les énergies 110
4.5.2 Modélisation du circuit test 112
4.5.2.1 Modèle VHDL- AMS du block POWER1 112
4.5.2.2 Modèle VHDL- AMS du block INTEG1 113
4.5.2.3 Modélisation SPICE des transistors MOS et composants de base 113
4.5.3 Simulation du circuit test Puissance & Energie 113
4.5.3.1 Netlist 113
4.5.3.2 Séquence de test 117
4.5.3.3 Puissance instantanée à 25°C (Figure 4-16) 118
4.5.3.4 Energie de transition instantanée à 25°C (Figure 4-17) 119
4.5.4 Comparaison des résultats de mesure et de simulation en fonction de la température 119
4.5.4.1 Mesure 119
4.5.4.2 Simulation 121

4.5.5 Discussion des résultats *122*
4.5.5.1 Courants mesurés et puissance calculée 122
4.5.5.2 Simulation 122
4.5.6 Utilisation du calculateur de puissance de Micron™ *123*
4.5.7 Conclusions et Commentaires sur la mesure et la simulation *125*
4.6 CONCLUSION 125

CHAPITRE 5 128

5 MODELISATION – SIMULATION DE SYSTEMES MULTI-TECHNOLOGIQUES 128

5.1 INTRODUCTION 128
5.2 MODELES COLORIMETRIQUES 128
5.2.1 VHDL-AMS pour la Colorimétrie *128*
5.2.1.1 Introduction 128
5.2.1.2 Les bases de la colorimétrie 130
5.2.1.3 Modélisation VHDL-AMS 132
5.2.1.4 Simulation du système colorimétrique 136
5.3 MODELISATION-SIMULATION ELECTRO-MECANIQUE 139
5.4 CONCLUSION 141

CONCLUSION GENERALE ET PERSPECTIVES 142

REFERENCES 145

PUBLICATIONS SCIENTIFIQUES ET DEPOT DE COPYRIGHT 151

PUBLICATIONS SCIENTIFIQUES 151
DEPOT DE COPYRIGHT DE LOGICIELS 152

ANNEXE 1 153

MODELE VHDL-AMS D'UN TRANSISTOR MOS NIVEAU 3 DE SPICE 153

ANNEXE 2 175

MODELE VHDL-AMS D'UNE CELLULE MEMOIRE DE TYPE EEPROM 175

ANNEXE 3 186

MODELE VHDL-AMS D'UN TRANSISTOR MOS : UNICELL 186

ANNEXE 4 197

MACRO-MODELE D'UN AMPLIFICATEUR OPERATIONEL MODE TENSION (AOV) MODELISE PAR PSPICE 197

ANNEXE 5 207

MODÈLES VHDL-AMS 207

INTRODUCTION GENERALE

Contexte général de l'étude

Contexte académique

Cette thèse s'inscrit dans le cadre de la recherche et du développement mené par l'équipe Modélisation Comportementale pour le Design (MCD) dans le Département Communications et Electronique (COMELEC) à l'ENST, autour de la conception VHDL-AMS de systèmes multi-technologiques.

La thèse d'Oussama Alali [OAL98], soutenue en mars 1998 et intitulée **"Modélisation VHDL-AMS et simulation SPICE"**, a permis de donner les bases d'une méthodologie de conception de systèmes multi-technologiques grâce à la construction d'un outil de simulation BVHDLA qui permet de simuler avec SPICE3f5 des modèles VHDL-AMS. Ces bases méthodologiques reposent sur la possibilité de modéliser tout ou une partie d'un système à différents niveaux : fonctionnel, comportemental (ou multi-technologique) et physique. De nombreux exemples analogiques ont été pris dans les domaines de l'électricité, de l'électronique, de la mécanique, de l'hydraulique, de la colorimétrie, de la thermique, ainsi que des combinaisons de ces différentes technologies, pour illustrer la faisabilité d'une opération.

La thèse de Jean Oudinot [OUD99], soutenue le 19 décembre 1999 est intitulée : **"Méthodologie de Conception d'ASIC mixtes avec VHDL-AMS"**. Compte tenu de l'absence de l'outil de synthèse analogique, l'approche des circuits analogiques ne peut pas imiter complètement celle des circuits numériques. Dans ce domaine, le niveau transistor prédomine la phase de conception. Désormais ces blocs de circuits peuvent être représentés par des modèles comportementaux. Il s'agit d'une démarche ascendante (*bottom-up)*. L'élévation de niveau d'abstraction des blocs analogiques apporte un gain radical en temps de simulation et rend la simulation système mixte possible. La combinaison des démarches *top-down* et *bottom-up* ouvre la porte à la conception de système mixte. Désormais il est possible de représenter tout bloc analogique et numérique dans un même environnement de simulation unifié. Ainsi le simulateur mixte de nouvelle génération permet de mixer des blocs de différentes natures à des différents niveaux d'abstraction du système. Le concepteur peut, selon le but recherché (conception ou vérification), choisir la combinaison des niveaux d'abstraction pour un compromis optimal entre la précision et le temps du CPU. La simulation du système mixte complet au niveau *top* est désormais

une réalité. Deux projets industriels se sont engagés dans la validation de cette nouvelle méthodologie.

Contexte industriel européen

Le projet européen SPARTE [SPA00] (Simulation based performance Assessment & Rating regarding Thermal & Electrical effects), s'est proposé de concevoir une méthodologie de conception de systèmes micro-électroniques dans un environnement thermique.

Objectif

Profitant des acquis et résultats des deux thèses citées auparavant et utilisant le contexte du projet européen industriel SPARTE qui constitue pour nous une base concrète de développement, notre travail porte sur la **contribution à l'élaboration de méthodologies et d'outils d'aide à la conception de systèmes multi-technologiques**, (dont font partie les Systèmes sur Puce (SoC, *System on Chip*)) avec des applications électroniques, électro-thermiques et multi-technologiques.

Structure du mémoire de thèse

Le mémoire de thèse, en plus de cette introduction générale et de la conclusion générale, est structuré en deux grandes parties : Méthodes & Outils et Applications

Première partie : Méthodes & Outils

Dans le 1er chapitre de cette première partie, chapitre intitulé : **Méthodes et Langages : vers l'utilisation du langage VHDL-AMS**, nous décrivons le cheminement qui nous a conduit à élaborer un outil spécifique à base de VHDL-AMS et de SPICE. Devant coopérer avec un consortium dans un projet européen, SPARTE (*Simulation based Performance Assessment & Rating regarding Thermal & Electrical effects*), nous avons proposé de travailler sur la partie méthodes et outils, et d'expérimenter nos propositions à l'aide de l'outil de conception de systèmes multi-technologiques que nous développions dans notre équipe et qui est devenu **VamSpiceDesigner**. Il s'agissait en effet de proposer des méthodes de vérification, par simulation, de la tenue de composants commerciaux (*COTS*) à la température dans la gamme militaire (-50°C ..+125°C) pour éviter les expérimentations longues et très coûteuses. Pour cela, quelles sont les méthodes à suivre, quel(s) langage(s) de modélisation et quel(s) simulateur(s) utiliser? On s'aperçoit que le simulateur SPICE ne peut pas entièrement répondre à ces questions, que le langage VHDL-AMS, de part ses potentialités, répond bien à nos attentes, et que notre idée de développer d'avantage VamSpiceDesigner s'affermit.

Le 2ème chapitre intitulé : **Outil : vers la construction de VamSpiceDesigner**, décrit effectivement l'élaboration de VamSpiceDesigner :

- Son historique,
- Les outils de développement analyseurs lexical et syntaxique,
- La construction du compilateur VamSpice permettant de traduire un modèle écrit en VHDL-AMS en modèle utilisable par SPICE
- L'adaptation à SPICE (version SPICE OPUS de l'Université de Ljubljana et XPICE de Georgia Tech Institute)
- L'adjonction d'une schématique (ELECTRIC) de CAO "facilement" adaptable à de la conception multi-technologique
- La possibilité d'une gestion de la modélisation – simulation à partir de la schématique.

Deuxième partie : Applications

Le 3ème chapitre est intitulé : **Modélisation – simulation de dispositifs MOS avec VHDL-AMS et avec SPICE**. Nous rappelons tout d'abord la modélisation SPICE du transistor MOS, puis présentons sa modélisation VHDL-AMS ainsi que celle de cellules EEPROM.

Le 4ème chapitre, intitulé : **Modélisation – simulation électro-thermique des mémoires FLASH et SDRAM**, est relatif à notre participation directe au projet européen SPARTE. Ce chapitre débute par une introduction sur la méthodologie générale de la simulation électro-thermique, méthodes directe et indirecte. De cette introduction est extraite la méthode utilisée dans le cadre de SPARTE : la méthode explicite à deux étapes qui sera appliquée aux mémoires FLASH et SDRAM prises comme démonstrateurs dans le cadre du projet. L'étude concernant la mémoire FLASH sera faite à l'aide de l'outil de simulation ADVanceMS de Mentor Graphics et celle relative à la mémoire SDRAM sera complètement menée avec VamSpiceDesigner. Cette dernière sera complétée par une étude comparative des résultats de mesure et de simulation en fonction de la température.

Le 5ème chapitre, intitulé : **Modélisation – simulation de systèmes multi-technologiques**, présente deux applications multi-technologiques avec VamSpiceDesigner.

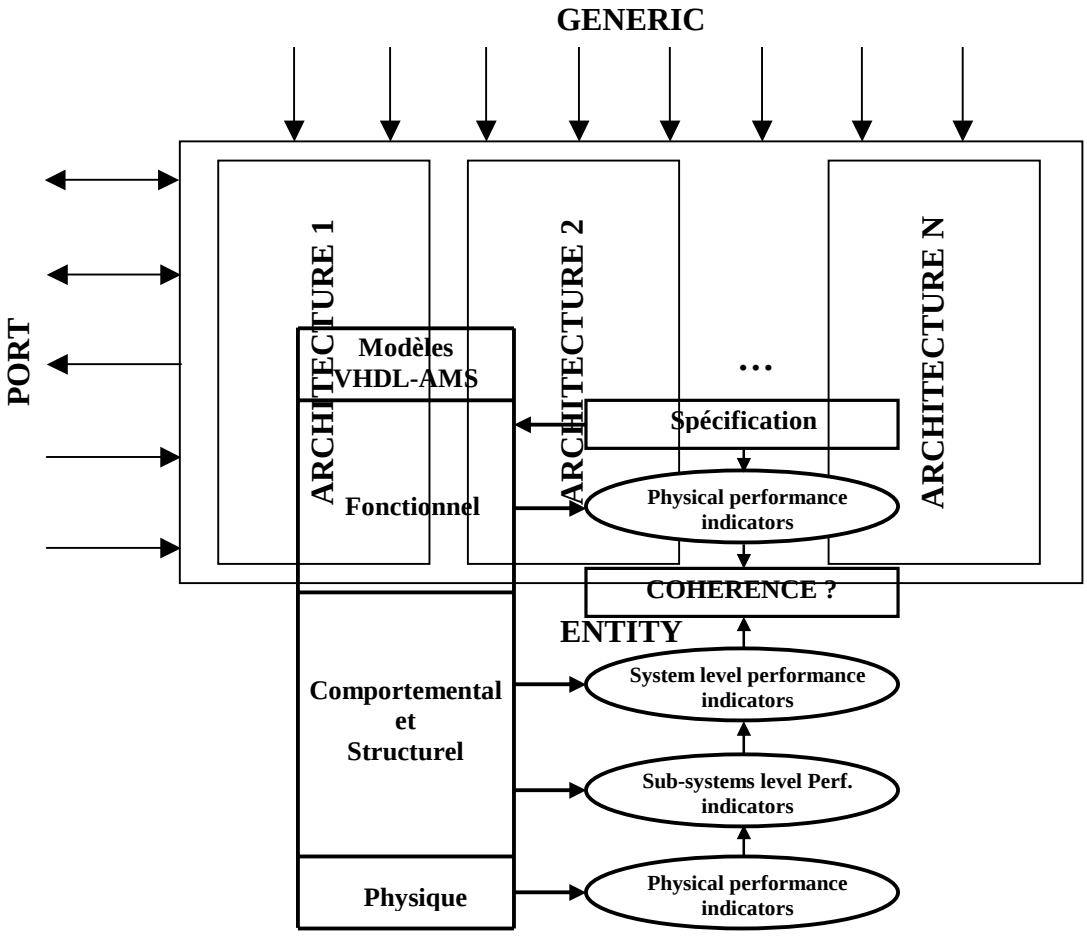
- L'une, plutôt inattendue, modélise et simule le comportement d'un **système colorimétrique**, basée sur l'analogie que nous avons trouvée entre la couleur et l'électricité. Le système colorimétrique considéré permet d'analyser, à partir de la schématique, le rendu de la couleur d'un objet coloré à travers le brouillard et à une certaine distance de l'objet.
- L'autre modélise et simule le comportement d'un système électro-mécanique tel que l'élévation de siège de voiture. Ce système contient des éléments électro-mécaniques tels qu'un moteur à courant continu, des engrenages, des convertisseurs de mouvements et un module pignon-crémaillère.

Première Partie

METHODES & OUTILS

1

Methodes et Langages : Vers l'Utilisation du Langage VHDL-AMS



Chapitre 1

1 METHODES ET LANGAGES : VERS L'UTILISATION DU LANGAGE VHDL-AMS

1.1 Introduction : Contexte de l'étude

Aujourd'hui, les fabricants de semi-conducteurs ne considèrent plus le marché militaire comme essentiel et le délaissent de plus en plus : des constructeurs comme MOTOROLA, et AMD se sont ainsi retirés de ce créneau. Cette évolution provient principalement du fait que les composants militaires ne représentent plus qu'une très faible part du marché de semi-conducteur (1%) et qu'ils demandent des contraintes de conception beaucoup plus importantes que les composants classiques de type commercial.

Que ce soit pour les puces, les ASICs, l'assurance de performances dans des conditions réelles d'utilisation n'est donc pas forcément garantie par les tests du fabricant. Le concepteur, tel MBDA, se sert ainsi de composants que le constructeur ne valide pas sur toute la plage d'utilisation. Pour palier à ces insuffisances, il se doit, au titre de sa maîtrise des processus, de mettre en place des techniques d'évaluation de l'aptitude à l'emploi de ces composants, dans des conditions déterminées par les spécifications, en particulier en température.

L'évolution en température d'un composant représente une des principales contraintes à son utilisation sur une carte. On distingue ainsi trois types de composants en fonction de la garantie fabricant en T°.

		Plage d'utilisation garantie	
Type de composant	Commercial	0°C	70°C
	Industriel	-40°C	85°C
	Militaire	-55°C	125°C

Tableau 1-1 : Plage d'utilisation garantie pour des composants.

MBDA intègre, depuis une dizaine d'années, dans certains de ses systèmes des composants commerciaux non spécifiés ni testés par les fabricants sur toute la plage de température de leurs conditions réelles d'application.

Afin de valider une carte employant de tels composants dans les conditions militaires, il est donc nécessaire de tester toutes leurs caractéristiques électriques pour s'assurer de leur

fonctionnement en gamme de température étendue. Ces opérations apparaissent très coûteuses car elles requièrent beaucoup de temps de manipulation et demandent des appareils de mesure très coûteux qu'il faut fréquemment renouveler pour pouvoir caractériser les nouveaux types de composants.

Pour limiter ces tests en laboratoire, MBDA a engagé plusieurs études visant à développer des méthodes et des outils de simulation qui devraient à terme se substituer aux tests ou du moins les limiter.

L'une des études menées par MBDA est associée au projet européen EUCLIDE CEPA2 qui est connu sous le nom de **SPARTE** : ***S**imulation based **P**erformance **A**ssessment & **R**ating regarding **T**hermal & **E**lectrical effects*. Cette thèse s'inscrit dans le cadre de ce projet.

Pour pouvoir répondre à l'étude, il est nécessaire de faire un point sur les méthodes et langages à utiliser.

1.2 Modèles et simulateurs

1.2.1 Modélisation et simulation

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

Deux façons permettent de représenter un modèle à temps discret :
- Par la communication des processus (signal),
- Par les équations booléennes.

Or, un modèle à temps continu ne peut être représenté que d'une seule façon :
- Par des équations analytiques ou mathématiques.

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

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 trois 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).

1.2.2 Limitations de SPICE

SPICE [NAG75] [WWW1][VLA94] (*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 qui 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 structurel 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 *Kirchoff* (loi des nœuds et loi des mailles). En ce qui concerne les flots de données (non-conservatif), une des façons de les représenter utilise le temps discret, or cette représentation n'est pas supportée par SPICE.

- **La 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 SPCIE 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.

1.2.3 Les différents types de langages

Nous pouvons distinguer principalement trois grandes familles de langages descriptifs : les langages de description logicielle, les langages de description structurelle et les langages de description matérielle appelés souvent HDL *: Hardware Description Langage*.

1.2.3.1 Les langages de description logicielle

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

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

Dans certains cas, le modèle décrit par un langage de description matérielle (VHDL, Verilog, etc.), est traduit en C/C++ avant d'être compilé ce qui offre à l'utilisateur un degré de liberté en ajoutant un co-langage comme le C/C++. Le langage principal ne permet pas parfois d'intégrer certains modèles à cause *de ces limitations*. Ainsi les co-langages servent à améliorer, affiner, et personnaliser les modèles à décrire. C'est le cas de HDL-A, MAST, ADVance MS™, etc.

1.2.3.2 Le langage de description structurelle (macro-modélisation)

1.2.3.2.1 Méthodologie de la macro-modélisation

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

Nous savons tous que le réseau ou circuit électrique décrit en *netlist* de type SPICE est analysé par le simulateur lui-même pour construire un système d'équations basées sur les lois de Kirchoff, donc, basées sur une loi de conservation d'énergie et sur les équations des composants. La macro-modélisation consiste à remplacer une partie d'un circuit ou d'un système par un autre modèle structurel plus simple (nombre de nœuds réduit) et plus près du circuit initial. Dans un autre sens, la macro-modélisation consiste à satisfaire des spécifications externes sans regarder la topologie initiale du circuit [POR03]. Le choix de l'un ou de l'autre, dépend du niveau d'abstraction. Le but essentiel de la macro-modélisation est de réduire la taille du circuit et ainsi de réduire le temps de simulation.

Les macro-modèles sont construits à partir d'un nombre réduit de composants idéaux. Les composants utilisés sont des composants primitifs de SPICE. Nous pouvons inclure des éléments passifs (résistance, capacité, etc.), des sources dépendantes et indépendantes, et des modèles non-linaire de bas niveau comme les diodes et les transistors bipolaires ou MOS de niveau 1 de SPICE [BOY74][TUR83].

Les sources contrôlées (sources dépendantes) sont des éléments idéaux qui permettent d'exprimer des relations mathématiques entre les courants et les tensions. SPICE définit quatre types de sources contrôlées [VLA94][QUA89][CON92]:

- Source de tension contrôlée par une tension (VCVS), de la forme :
  $V_E = EV_C$ (linaire) ,
  $V_E = e(V_C)$ (non-linaire).
- Source de tension contrôlée par un courant (CCVS), de la forme :
  $V_H = HI_C$ (linaire) ,
  $V_H = h(I_C)$ (non-linaire).
- Source de courant contrôlée par un courant (VCCS), de la forme :
  $I_G = GI_C$ (linaire) ,
  $I_G = g(I_C)$ (non-linaire).

- Source de courant contrôlée par une tension (CCCS), de la forme :
  $I_G = GI_C$ (linaire),
  $I_G = g(I_C)$ (non-linaire).

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 1-1) et elle est de la forme [CON92]:

$$V = N.V_T.\ln\left(\frac{I_D}{I_S} - 1\right) \qquad (1.1)$$

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 1-1 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 1-1). 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 [BOY74].

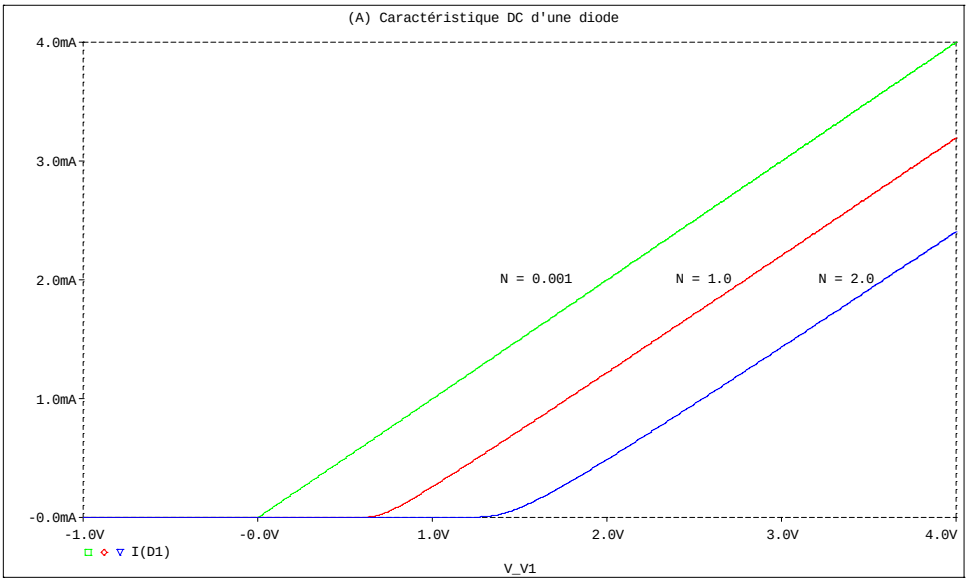


Figure 1-1 : Caractéristique d'un modèle SPICE d'une diode avec différentes valeurs de coefficient d'émission (N).

La méthode de macro-modélisation consiste à utiliser trois blocs de base composés par (Figure 1-2):

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.

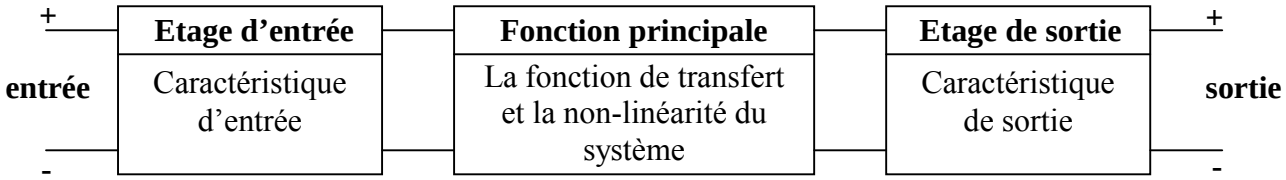


Figure 1-2 : Méthodologie de base de la macro-modélisation [CON92].

Pour appliquer cette méthode, nous avons développé un macro-modèle générique d'un amplificateur opérationnel mode tension (AOV) à deux pôles réels. Pour tester les performances de cet amplificateur, nous l'avons simulé en l'incluant dans plusieurs circuits pour montrer que les performances appliquées sont les mêmes que les performances obtenues (Annexe 4).

1.2.3.2.2 Avantages et inconvénients

L'avantage majeur de la macro-modélisation, 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 macro-modélisation, nous pouvons alors réaliser un grand nombre de fonctions : fonctions mathématiques, filtres, détecteurs de pics, détecteurs de phase, modulateurs d'amplitude et de largeurs d'impulsions, oscillateur, convertisseur fréquence-tension, comparateurs, amplificateurs et boucle à verrouillage de phase. Tous ces applications sont décrites dans [CON92].

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

- La non-linérarité 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.

1.2.3.3 Les langages de description matérielle

L'utilisation des langages de description matérielle pour la conception et la synthèse des circuits digitaux (VLSI) réduit les temps de conception. Nous pouvons citer quelques exemples des langages les plus connus comme VHDL et Verilog. De ce fait, l'idée des langages de description matérielle pour la conception des circuits analogiques et mixtes est venue de ces langages numériques et après de nombreuses tentatives. Nous pouvons cité l'exemple de HDL-A de Mentor

Graphics® qui est associé à Eldo, le langage MAST d'Analogy® puis Synopsys® qui est associé à Saber® HDL, et le langage SpectreHDL de Cadence® qui est associé à Spectre®. Ces langages sont appelés des **langages propriétaires** car non issus de standards.

Cependant, de nouvelles normes sont apparues, à la base de ses langages propriétaires (Figure 1-3 et 1-4). Ces normes sont des extensions des langages existant comme par exemple le VHDL-AMS qui est une extension de VHDL, et le Verilog-AMS qui est l'extension de Verilog. Ces nouveaux langages sont destinés à la modélisation analogique et mixte mais il existe des langages qui sont des sous-ensembles des langages principaux, nous citons par exemple le VHDL-A et Verilog-A (Figure 1-3) qui traitent uniquement la partie analogique.

```
module amp(sigin, sigout);
input sigin;
output sigout;
electrical sigin, sigout;
parameter real gain = 2.0;
parameter real sigin_offset = 0;
 analog
    V(signout) <+ gain * (V(sigin) - sigin_offset);
endmodule


module cap (vp, vn);
inout vp, vn;
electrical vp, vn;
parameter real c = 0;
analog
    I(vp, vn) <+ ddt ( c * V(vp, vn) );
endmodule
```

Figure 1-3 : Un modèle d'amplificateur opérationnel et un modèle d'une capacité décrits en Verilog-A.

```
module amp(sigin, sigout) (gain, sigin_offset)
node sigin, sigout;
parameter real gain=2.0;
parameter real sigin_offset = 0;
{
  analog
    val(sigout) <- gain * (val(sigin) - sigin_offset);
}


module cap(vp, vn) (c)
node (V, I) vp, vn;
parameter real c = 0;
{
  analog
    I(vp, vn) <- dot ( c * V(vp, vn) );
}
```

Figure 1-4 : Un modèle d'amplificateur opérationnel et un modèle d'une capacité décrits en SpectreHDL.

1.3 Brève présentation du langage VHDL-AMS

1.3.1 Introduction

Le langage VHDL (*Very High Speed Integrated Circuits Hardware Description Langage*) est un puissant langage de description des circuits électroniques numériques. Avec le VHDL, il est possible de simuler et de synthétiser des circuits numériques pour différentes technologies. Toutes entités (ENTITYs) déjà créées sont archivées dans une librairie de travail (*Work Library*) pour être réutilisées ou modifiées plus tard. Le gros avantage en matière de productivité est lorsqu'une librairie comporte beaucoup de composants simples prêts à être intégrés à des systèmes plus complexes.

La première standardisation de ce langage a eu lieu en 1987. En 1993, une version améliorée a été aussi standardisée. Elle devait permettre d'intégrer des systèmes analogiques mais beaucoup de travail devait être réalisé pour y parvenir. Un groupe de travail spécifique a donc été formé pour développer une extension au langage qui pourrait permettre la description et la simulation de systèmes mixtes analogiques. Cette version étendue a pour nom VHDL-AMS (AMS : *Analog and Mixed Signal*) et a été standardisée sous le nom IEEE – VHDL 1076.1.

1.3.2 Modélisation comportementale analogique

Les aspects continus des comportements des parties de systèmes visés par la norme IEEE VHDL 1076.1 peuvent être décrits par un système d'équations différentielles et algébriques (DAE) avec le temps comme variable indépendante. Ces équations sont de la forme : Equation 1.2

$$F\left(x, \frac{dx}{dt}, t\right) = 0 \tag{1.2}$$

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

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

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

Il est nécessaire de faire la différence entre deux types de EDA (Figure 1-5):

- Non-conservatif,
- Conservatif.

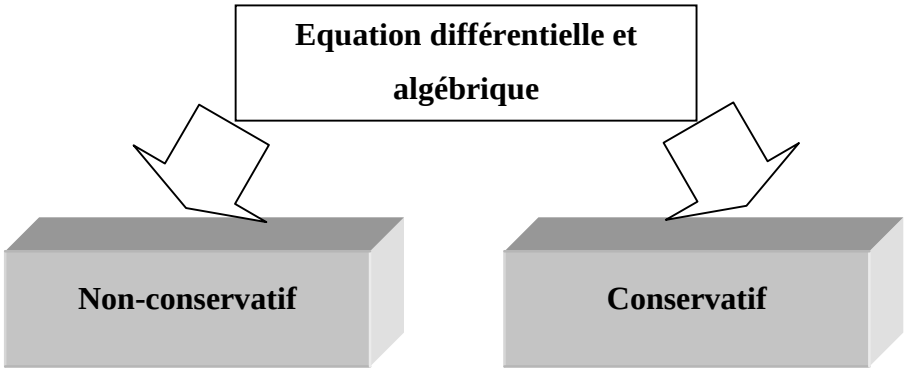


Figure 1-5 : Types des équations différentielles et algébriques.

1.3.2.1 Les systèmes analogiques conservatifs

Dans un système conservatif, les équations différentielles et algébriques contiennent des inconnues définies explicitement et sont régies par les lois de la physique et les lois de conservation d'énergie. Les circuits électriques sont de bons exemples de systèmes à temps continu et sont gouvernés par des ensembles d'équations conservatives (lois de Kirchoff). De même les systèmes mécaniques sont gouvernés par des équations conservatives obéissant aux lois de Newton et aux lois de la cinématique. Les structures et les formalismes des systèmes conservatifs sont exploités dans VHDL-AMS par des instructions telles que *terminal* et *nature* qui seront décrites dans les prochaines sections. Pour cela, également on cite deux exemples d'équations d'un système conservatif : Equation 1.3 et 1.4

$$\text{Vitesse} = \frac{d\,(\text{déplacement})}{dt} \qquad \text{et} \qquad V = R\,I \qquad (1.3) \text{ et } (1.4)$$

1.3.2.2 Les systèmes analogiques non-conservatifs

Le comportement dynamique du système non-conservatif est décrit à l'aide d'une DAE, mais il n'est pas nécessaire qu'il soit à conservation d'énergie. Cette terminologie n'implique pas les équations différentielles et algébriques, car les inconnues ne sont pas définies et les équations ne sont pas produites en utilisant les constructions de VHDL-AMS qui fournissent explicitement et systématiquement identifiant tout en utilisant les lois de physique et la conservation d'énergie. VHDL-AMS exploite cela par les quantity libres qui seront décrit dans les prochaines sections.

Pour illustrer les systèmes analogiques non-conservatifs et pour montrer la capacité de VHDL-AMS à décrire les équations mathématiques, on peut citer un exemple d'équation différentielle du premier ordre d'un système non-conservatif : Equation 1.5

$$\frac{dU(t)}{dt} = C_0\,U(t) \qquad (1.5)$$

1.3.3 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 *PORT*s 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 1-6)

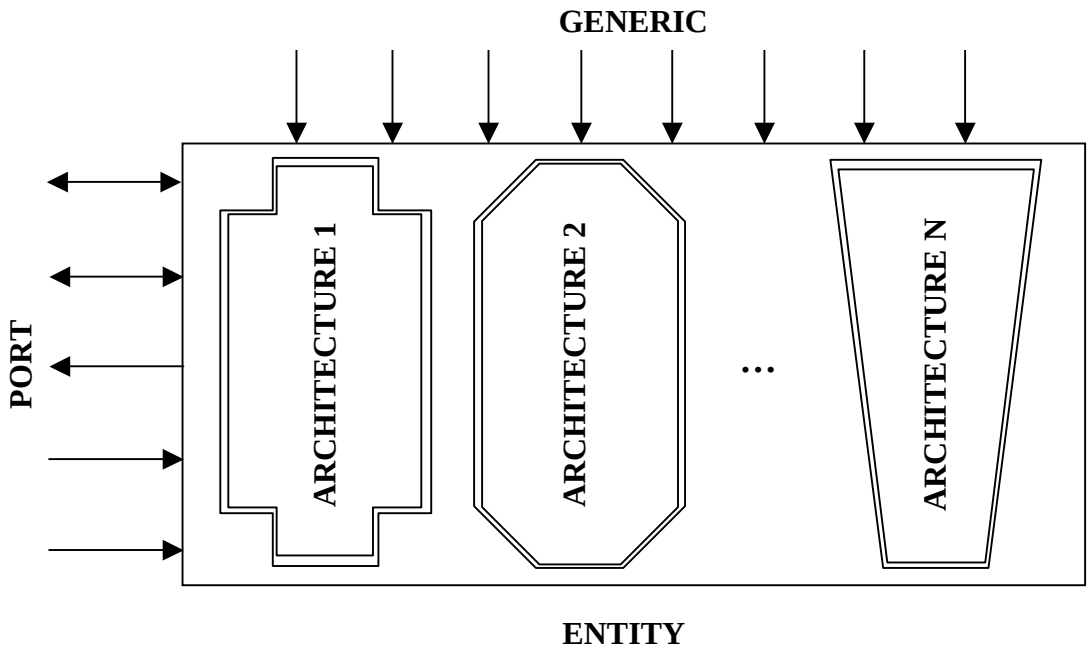


Figure 1-6 : Structure d'un modèle VHDL-AMS.

Syntaxe [IEE98]:

```
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...
```

1.3.4 Classes d'objets

1.3.4.1 QUANTITYs

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 <quatity_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'dot* pour désigner la quantité du même type que *Q* qui représente la dérivée de *Q* en fonction du temps. On note aussi *Q'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'dot'dot* (ordre 2). Il en est de même pour les intégrales avec *Q'Integ'Integ* (ordre 2).

1.3.4.2 TERMINALs et NATUREs

Les ***TERMINAL***s 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 quatity*) 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 sont de nature simple : Tableau 1-1

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 1-2 : Effort et effet pour différents domaines.

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.

1.3.5 Déclarations simultanées

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 nouvelle.

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];
```

L'exemple suivant présente les nouveaux concepts du langage VHDL-AMS :

```
ENTITY limiter IS
        GENERIC(  Gain    : REAL := 1.0;
                     CONSTANT  Limit   : REAL
                  );
        PORT(  TERMINAL inp,inm: in Electrical;          -- terminaux d'entrée
               TERMINAL p  , m : out Electircal          -- terminaux de sortie
             );
```

```
BEGIN
        ASSERT Gain>=0.0 REPORT "ERREUR : Le Gain doit être positif";
END ENTITY limiter;

-- architecture simultanée
ARCHITECUTRE simult OF limiter IS
        QUANTITY Vin  across                inp TO inm;
        QUANTITY V    across I through    p    TO   m;
BEGIN
        IF ( (Gain * Vin ) > Limit ) USE
            V == Limit;
        ELSIF ( (Gain * Vin) < (-Limit) ) USE
            V == - Limit;
        ELSE
            V == Gain * Vin;
        END USE;
END ARCHITECTURE simult;
-- architecture procédurale
ARCHITECTURE proc OF limiter IS
        QUANTITY Vin     across                  inp TO inm;
        QUANTITY V       across I   through      p  TO  m;
BEGIN
        proced :
        PROCEDURAL  IS
                VARIABLE Vout : VOLTAGE;
        BEGIN
                Vout := Gain * Vin;
                V      := Vout;
                IF    ( Vout  >   Limit )   THEN
                        V := Limit;
                ELSIF ( Vout  < (- Limit) ) THEN
                        V := -Limit;
                END IF;
        END PROCEDURAL;
END proc;
```

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 (***TOLERANCE***[1]) qui sont 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.

Syntaxe :

```
SUBTYPE <variable_1> IS REAL TOLERANCE  "default_variable_1";
QUANTITY <variable_q_1> : variable_1;
```

[1] La norme ne précise pas comment un outil utilise les groupes de tolérance.

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 discontinué 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;          -- branch quantities to ground
BEGIN
      IF   S = '0'     USE
          V == 0.0;                           -- low output
      ELSE
          V == Vhigh;                         -- high output
      END USE ;
      BREAK  ON  S;                           -- prévision de discontinuité
END ARCHITECTURE arch_dac;
```

1.4 Apports de VHDL-AMS

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

- **La modélisation mixte :**
  - *Quantity* et *Terminal s*ont 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 évènement discret *signal* pour représenter la modélisation mixte d'un système. *Terminal* représente les nœuds des branches *across* or *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) [HER02] 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 ***Q****'dot* 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 *Kirchoff* 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, SystemC, 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* et *through 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.

1.5 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 1-7.

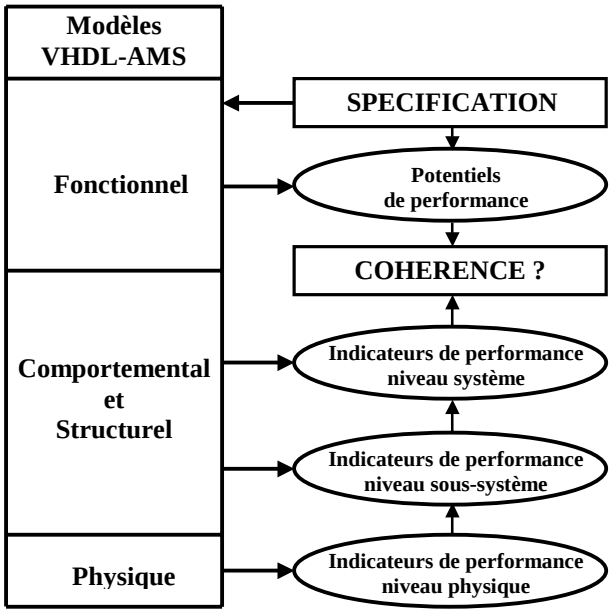


Figure 1-7 : Méthodologie de conception *top-down* et *bottom-up* avec VHDL-AMS.

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 [CHA00]. 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.

Les manœuvres de navigation peuvent être obtenues en pointant sur une boîte pour faire apparaître les autres blocs de niveau d'abstraction inférieur ou supérieur.

1.6 Conclusion

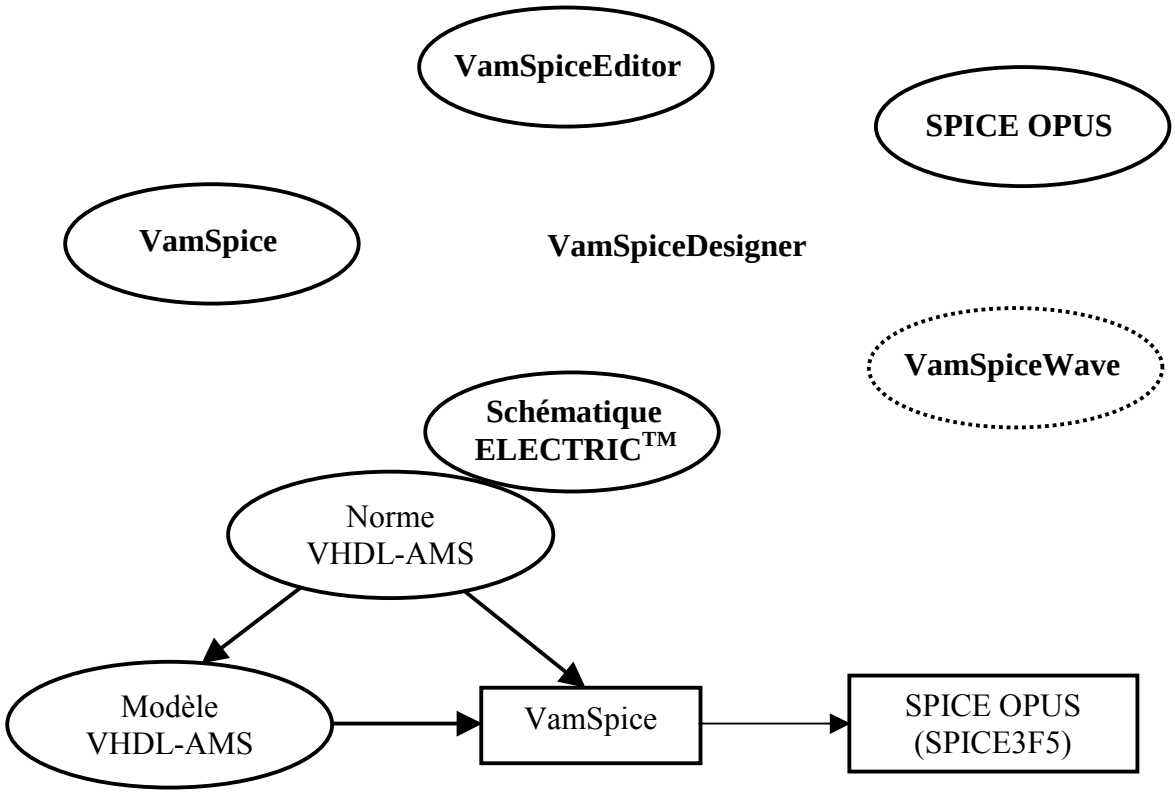
Dans ce chapitre, nous avons décrit le cheminement qui nous a conduit à élaborer un outil spécifique à base de VHDL-AMS et de SPICE.

Pour ce là, nous avons vu le but essentiel du projet SPARTE qui propose des méthodes de vérification, par simulation, de la tenue de composants commerciaux (*COTS*) à la température dans la gamme militaire (-50°C..+125°C) pour éviter les expérimentations longues et très coûteuse. Pour pouvoir répondre à cette étude, nous avons fait le point sur les méthodes et langages à utiliser. Nous avons vu 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, de part ses potentialités, répond bien à nos attentes.

Pour mettre ces méthodes en oeuvre, nous proposons de développer un outil de conception de systèmes multi-technologiques pour vérifier les méthodes proposées. Cet outil est appelé **VamSpiceDesigner**.

2

OUTIL : VERS LA CONSTRUCTION DE VAMSPICEDESIGNER



Chapitre 2

2 OUTIL : VERS LA CONSTRUCTION DE VAMSPICEDESIGNER

2.1 Historique et introduction

2.1.1 Historique : BVHDLA

Compte tenu de l'orientation que le laboratoire MCD (initialement Modélisation Caractérisation des Dispositifs) de l'ENST prenait : à savoir passer de la modélisation physique de dispositifs à la modélisation de systèmes, nous devions considérer d'autres langages et méthodes de modélisation. HDL-A [HDL94] d'ANACAD®, l'un des premiers langage de description matérielle, nous a permis de découvrir les potentialités d'un langage de modélisation comportementale purement analogique. Donc, l'idée est venue d'utiliser le simulateur SPICE comme solveur d'équations, en lui associant un traducteur (compilateur).

Pour tester la faisabilité de cette opération, le laboratoire a commencé par associer le langage HDL-A de ANACAD® à SmartSpice™ de SILVACO au moyen d'un traducteur (SVHDLA) qui permettait de passer d'un modèle HDL-A en un modèle-interpréteur reconnu par SmartSpice™. L'interpréteur de SmartSpice™ facilite bien les choses, car les éléments générés par SVHDLA étaient introduits dans une structure de modèle comportant peu de fichiers.

Mais l'inconvénient avec SmartSpice était que son interpréteur n'autorisait pas toutes les possibilités de SPICE et que c'était un produit commercial. Le laboratoire a donc remplacé SmartSpice par SPICE3F5, la version générique de SPICE de l'Université de Berkeley. Pour faire le lien entre le langage et SPICE3F5, le laboratoire a conçu un nouveau traducteur BVHDLA [OAL98] à base du langage HDL-A. Une fois que le Manuel de Référence de VHDL-AMS (fin 1996) a été mis en place, BVHDLA a été adapté en y introduisant progressivement les éléments essentiels du langage.

Le principe de BVHDLA est d'introduire dans la structure des fichiers d'un modèle SPICE les modifications nécessaires (équations du modèle VHDL-AMS, ses entrées et sorties,…) et recompiler le tout, c'est à dire les fichiers générés par BVHDLA et les fichiers de SPICE, pour que la simulation de ce modèle SPICE modifié corresponde à la simulation du modèle VHDL-AMS.

2.1.2 Introduction

La méthode proposée (cf. §2.1.1) pour simuler un modèle VHDL-AMS avec SPICE est très lourde, car on doit générer plusieurs fichiers (une vingtaine de fichiers) pour constituer un modèle de type SPICE et recompiler le tout pour générer un exécutable SPICE qui reconnaisse ce modèle VHDL-AMS.

Pour cela, nous allons montrer dans ce chapitre comment améliorer le compilateur BVHDLA et l'environnement de simulation. Nous élaborons une méthodologie hiérarchique faisant appel à de la modélisation, de la simulation et un traitement schématique. Cette méthode nous a amené à fabriquer notre propre outil de simulation. Cet outil est appelé VamSpiceDesigner.

2.2 Compilateur VamSpice

2.2.1 Principe du compilateur

Conceptuellement, un compilateur opère en phases, chacune d'elles transformant le programme source d'une représentation en une autre. Une décomposition typique d'un compilateur est présentée dans la figure 2-1.

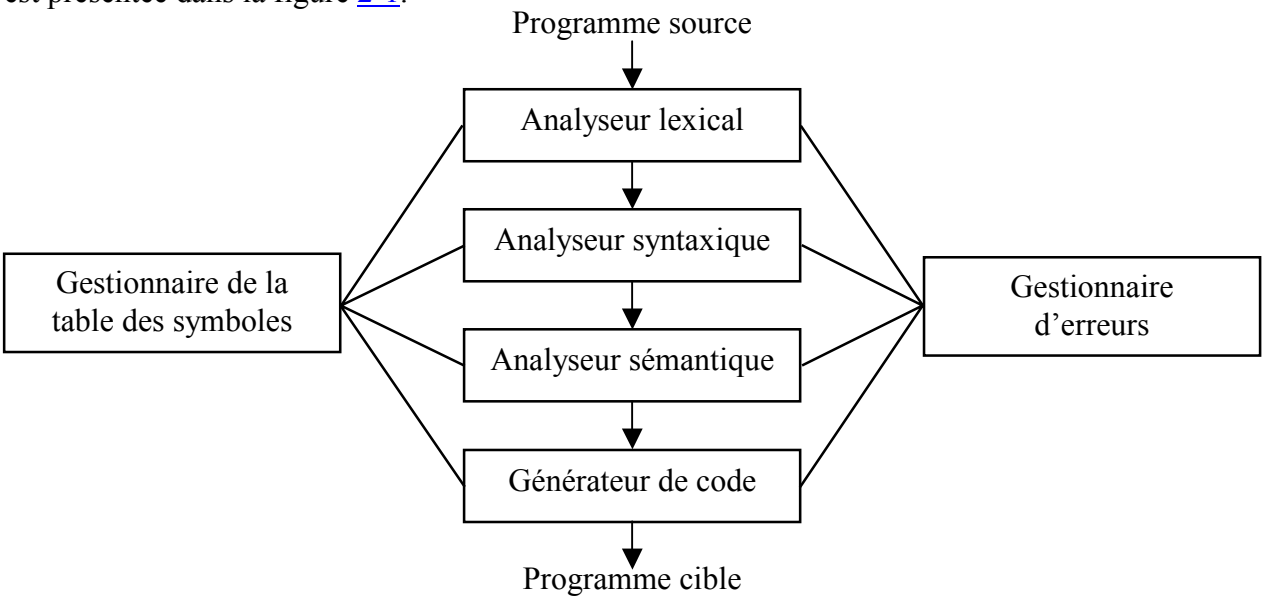


Figure 2-1 : Phases d'un compilateur

La représentation interne au compilateur du programme source évolue au fur et à mesure de la traduction. Nous illustrons ces différentes représentations en considérant la traduction de l'instruction VHDL-AMS suivante :

$$\text{position} == \text{initiale} + \text{vitesse} * 60.0 \qquad (2.1)$$

La figure 2-3 montre la représentation de cette instruction à l'issue de chaque phase.

2.2.1.1 Phases d'analyse

L'analyseur lexical lit les caractères formant le programme source et les groupe en un flot d'unités lexicales, dont chacune représente une suite de caractères formant un tout logiquement cohérent, comme un identificateur, un mot clé (**if**, **then**, **break**, ….), un caractère de ponctuation, ou un opérateur composé de plusieurs caractères comme « == ». La suite de caractères composant une unité lexicale est appelée son *lexème*.

Certaines unités lexicales se verront attacher une « valeur lexicale ». Par exemple, quand on rencontre un identificateur comme **vitesse**, l'analyseur lexical ne se contente pas de produire une unité lexicale, par exemple **id**, mais il entre en outre le lexème **vitesse** dans la table des symboles, s'il n'y est pas déjà. La valeur lexicale associée à cette occurrence de **id** pointera vers l'entrée de **vitesse** dans la table des symboles.

Dans la suite, nous emploierons $\mathbf{id_1}$, $\mathbf{id_2}$, $\mathbf{id_3}$ au lieu de **position**, **initiale** et **vitesse** respectivement, de façon à mettre en valeur le fait que la représentation interne d'un identificateur est différente de la suite de caractères qui le forme. La représentation de (2.1) à l'issue de l'analyse lexicale est donc suggérée par :

$$id_1 == id_2 + id_3 * 60.0 \qquad (2.2)$$

La phase d'analyse syntaxique est appelée l'*analyse hiérarchique* (ou quelquefois analyse grammaticale). Elle consiste à regrouper les unités lexicales du programme source en structures grammaticales qui seront employées par le compilateur pour synthétiser son résultat. En général, ces phases grammaticales sont représentées par un arbre syntaxique comme celui de la figure 2.2.

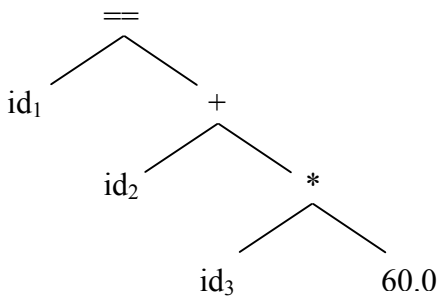


Figure 2-2 : La structure de données

La phase d'analyse sémantique contrôle si le programme source contient des erreurs sémantiques et collecte des informations de type destinées à la phase de production de code. Elle utilise la structure hiérarchique déterminée par la phase d'analyse syntaxique pour identifier les opérateurs et les opérandes des expressions, ainsi que les instructions.

2.2.1.2 Production de code

A l'issue de l'analyse syntaxique et de l'analyse sémantique, la phase finale d'un compilateur est la production de code cible qui consiste normalement en code machine ou en code en langage d'assembleur.

Dans notre cas la production de code se fera en langage C. Ce sujet est traité en détail dans 2.2.2.

2.2.1.3 Gestion de la table des symboles

Une fonction essentielle d'un compilateur est d'enregistrer les identificateurs utilisés dans le programme source et de collecter de l'information sur divers attributs de chaque identificateur. Ces attributs fournissent de l'information concernant, par exemple, son type, son emplacement, ses paramètres quand il s'agit d'une procédure ou d'une fonction.

2.2.1.4 Détection et compte rendu des erreurs

Les phases d'analyse syntaxique et d'analyse sémantique traitent en général une grande part des erreurs détectables par le compilateur. La phase lexicale peut détecter des erreurs quand les caractères restant à lire ne peuvent former aucune unité lexicale du langage. Les erreurs dues au fait

que le flot d'unités lexicales n'est pas conforme aux règles structurelles (la syntaxe) du langage sont détectées par la phase d'analyse syntaxique. Au cours de l'analyse sémantique, le compilateur s'efforce de détecter des constructions ayant une structure grammaticale correcte.

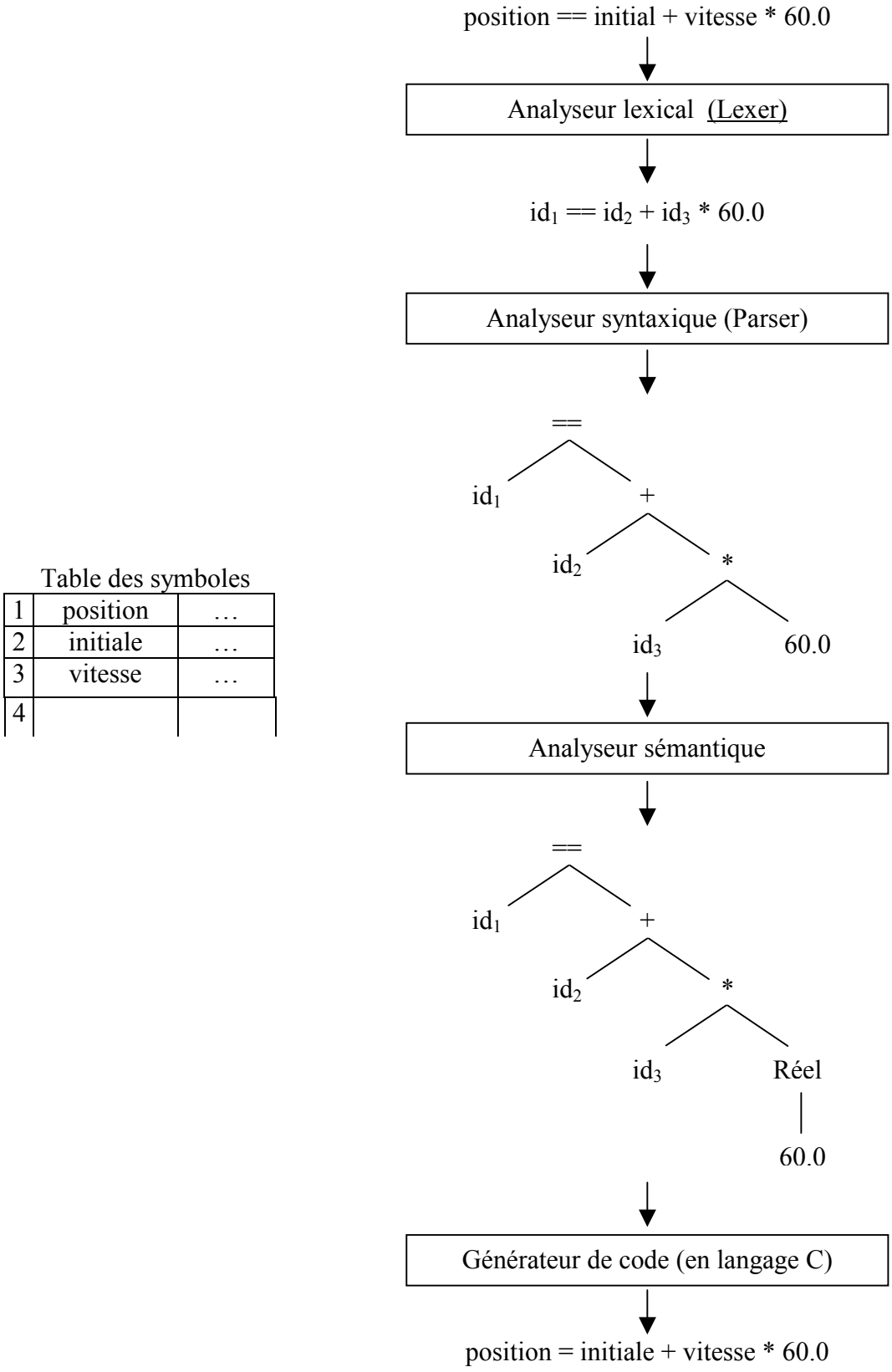


Figure 2-3 : Traduction d'une instruction VHDL-AMS en langage C.

2.2.2 VamSpice (VamSpiceCompiler)

2.2.2.1 Principe

VamSpice (**V**HDL-**AMS**/**SPICE**) est un compilateur qui traduit un modèle VHDL-AMS utilisable par SPICE OPUS, la figure 2-4 montre ce passage. Le traducteur génère principalement deux fichiers d'interface **cfunc.c** et **ifspec.c**. Une fois ces deux fichiers compilés, on peut les relier à SPICE OPUS.

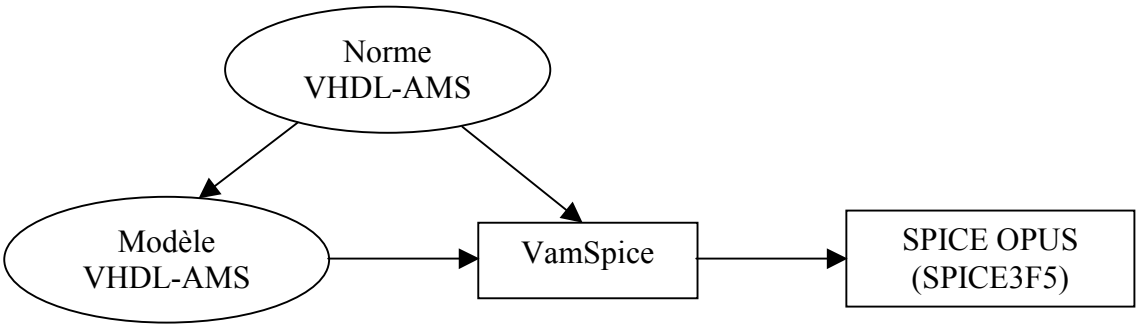


Figure 2-4 : De VHDL-AMS à SPICE OPUS

2.2.2.2 Méthodologie

Pour créer le traducteur VamSpice, nous avons utilisé le langage commercial de programmation des compilateurs YACC++™ [YAC96] (***Y**et **A**nother **C**ompiler **C**ompiler and the Language Objects Library*). L'avantage de ce compilateur est de pouvoir contenir dans un même fichier l'analyse lexical « *lexer* » et l'analyse syntaxique « *parser* ».

Un fichier **vamspice.yxx** contient une partie de la « grammaire » de VHDL-AMS [AMS99]. La partie lexicale contient les mots clés du langage VHDL-AMS, les caractères spéciaux tels que fin de fichier, retour chariot, tabulation etc., les définitions des réels à virgule flottante, les entiers, les identifications, les commentaires et les opérateurs arithmétiques et logiques. Cet ensemble d'unités lexicales est appelées symboles ***terminaux*** (***pattern***). La partie « *parser* » contient un ensemble de productions (***production rule***) où chaque production est constituée d'un ***non-terminal***, appelé partie gauche de production, d'une flèche → ou le signe ::= désigne une règle syntaxique, et d'une suite d'unités lexicales et de non-terminaux appelée partie droite de la production.

On prendra par convention pour la suite **:** un nom ***en italique*** dénote un non-terminal et ***en gras*** un terminal ou une unité lexicale. Le point virgule « ; » désigne la fin d'une production et la barre verticale « | » est une alternative.

Par exemple, une instruction en VHDL-AMS exprimée dans le format standard BNF [JRL92] [AHO00] (acronyme de *Backus-Naur Form*) est de la forme :

simultaneous_if_statement ::= **IF** *boolean_expression* **USE** *simultaneous_statement_part*
ELSE *simultaneous_statement_part* **END USE**

boolean_expression ::= "(" *simple_expression boolean_operator simple_expression* ")"
simple_expression ::= *identifier*
| …

boolean_operator ::= ">"
| ">="
| **AND**
| **OR**
| "=" …

devient en Yacc++ :

simultaneous_if_statement : **IF** *boolean_expression* **USE** *simultaneous_statement_part*
ELSE *simultaneous_statement_part* **END USE**;
boolean_expression : "(" *simple_expression boolean_operator simple_expression* ")" ;
simple_expression : *identifier*
| … ;
boolean_operator : ">"
| ">="
| **AND**
| **OR**
| "=" … ;

On effectue la construction descendante d'un arbre syntaxique en partant de la racine, étiquetée par l'axiome, et en réalisant de manière répétitive les étapes ci-dessous (Figure 2-5).

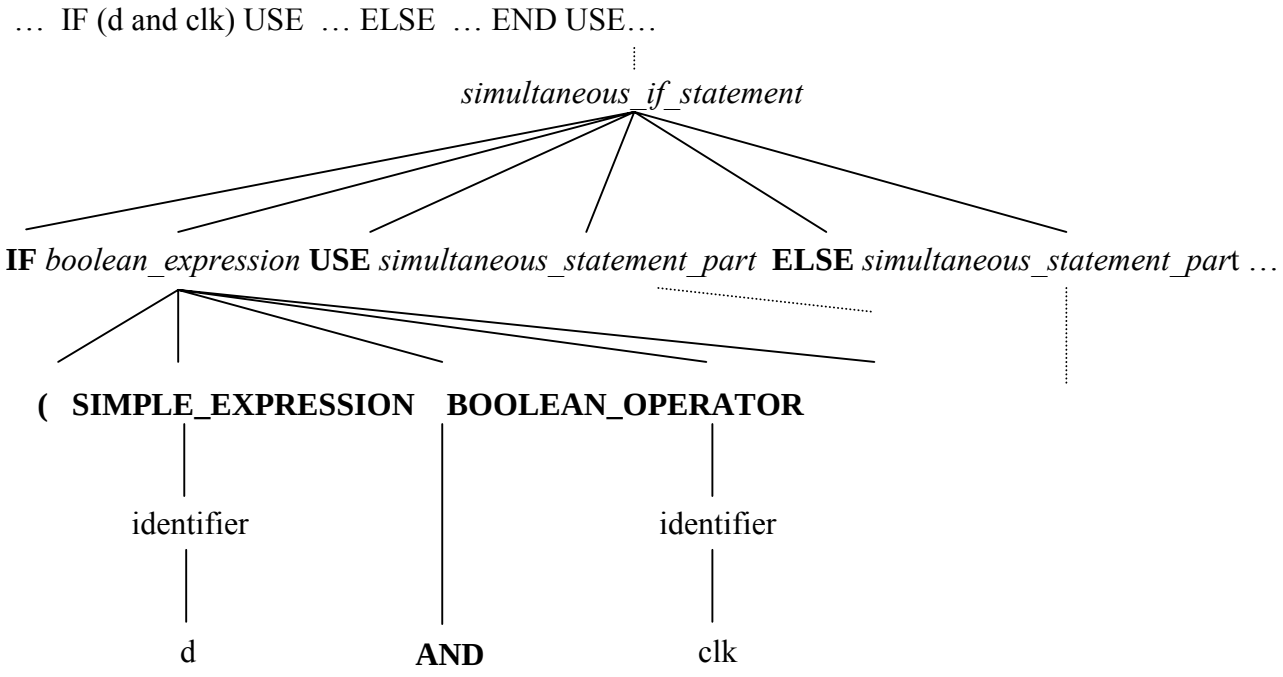


Figure 2-5 : Construction descendante d'un arbre syntaxique

Enfin, l'analyseur syntaxique est à l'origine de la construction d'un enregistrement de données, appelée **arbre abstrait** [AHO00], à partir duquel sera généré le code de sortie. L'arbre abstrait est une forme condensée de l'arbre syntaxique (Figure 2-6) .

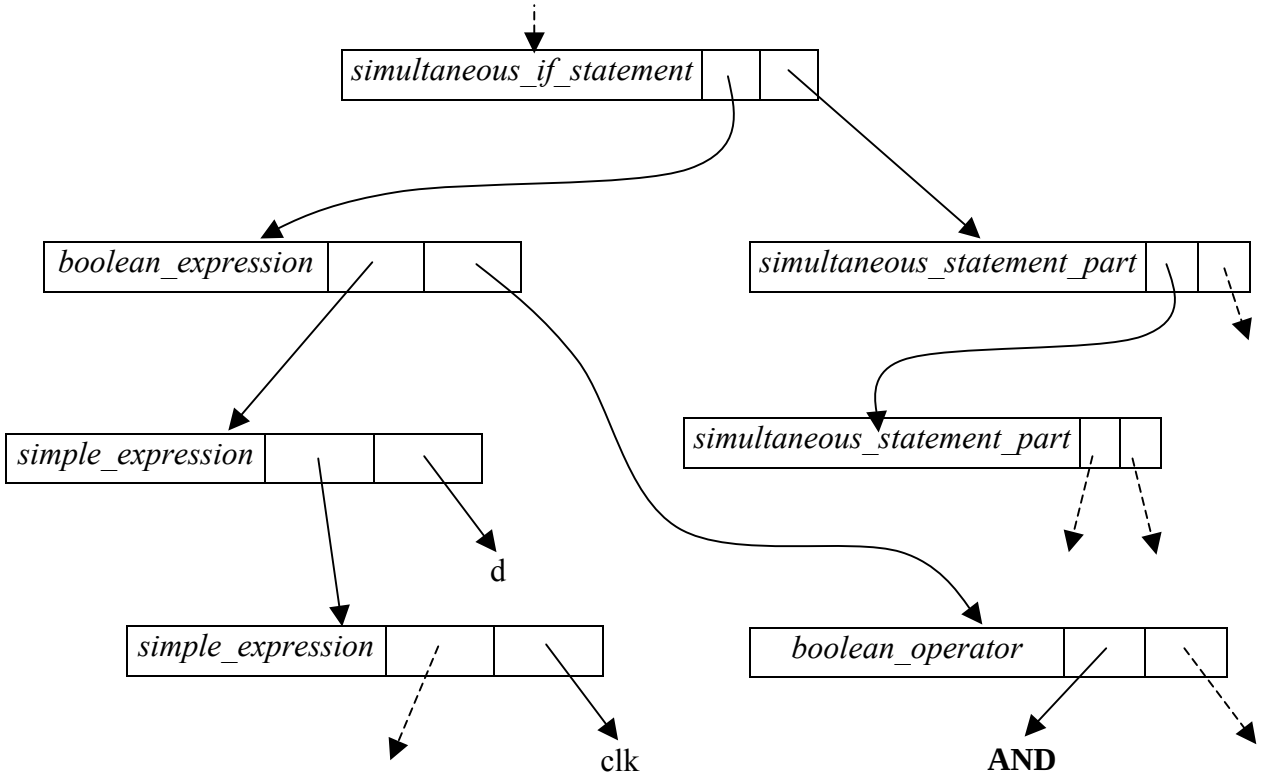


Figure 2-6 : Construction descendante d'un arbre abstrait

Pour manipuler une telle structure, des *actions* sont associées aux règles grammaticales appelées **actions sémantiques**. Ces actions sont exécutées lorsque la règle est reconnue. Pour cela, nous avons défini quelques fonctions d'action qui nous aideront à extraire les données utiles d'un modèle VHDL-AMS (**Add_x**, **concat_3**, …).

simultaneous_if_statement : **IF** *boolean_expression* **USE** *simultaneous_statement_part*
{ actions sémantiques }
ELSE *simultaneous_statement_part*
{ actions sémantiques }
END USE;

Nous compilons le fichier du grammaire **vamspice.yxx** avec l'exécutable de Yacc++ et nous obtenons 2 fichiers C++. Ces deux fichiers sont liés avec les fichiers des fonctions d'actions sémantiques pour obtenir le fichier exécutable de notre compilateur.

2.2.2.3 Création d'un modèle VHDL-AMS

Nous utilisons **vamspice.exe** pour compiler un modèle VHDL-AMS. Le résultat de la compilation est stocké dans des fichiers de type C (cfunc.c et ifspec.c). Nous compilons ces deux fichiers avec les outils et les bibliothèques de GNU et nous obtenons après le lien avec la bibliothèque de SPICE OPUS un fichier DLL (*Dynamic Link Library*) ou tout simplement un fichier de type .CM : *code model*. En même temps, est mis à jour un fichier appelé *workfile.ini* qui représente la bibliothèque de travail par défaut (*default work library*) et qui contient les chemins et les noms des modèles qui seront utilisés par SPICE OPUS pour la simulation.

Vamspice.exe est appelé directement en ligne de commandes (ou à l'invite de commandes du DOS) ou par l'éditeur de modèle comme étant une commande interne. La syntaxe d'utilisation est la suivante :

Syntaxe : vamspice.exe [<options> <nom_de_fichier>] | [<nom_de_fichier> <options>]

les <options> sont :

-compile| -c
-build | -b
-lib | -l
-setlib | -sl
-dir | -d
-del_cm | -dcm
-help | -h

Arguments des options :

-compile| -c <nom_du_modele.vhd>
: Compiler le modèle VHDL-AMS sans faire le lien.

-build | -b < nom_du_modele.vhd>
: Compiler le modèle VHDL-AMS et faire le lien pour générer un fichier .DLL (ou .CM).

-lib | -l <nom_librarie.ini>
: Créer une bibliothèque de travail (*work library*) par défaut se crée workfile.ini.

-setlib | -sl <nom_librarie.ini>
: Initialiser la bibliothèque de travail.

-dir | -d <nom_librarie.ini>
: Afficher le continu de la bibliothèque.

-del_cm | -dcm <nom_de_fichier.vhd>
: Effacer un modèle VHDL-AMS d'une bibliothèque.

-help | -h
: Afficher la syntaxe d'appel de vamspice.exe et les arguments des options.

Exemples :

vamspice.exe –b resistance.vhd ↵
VamSpice compile et lie le contenu du modèle VHDL-AMS « resistance.vhd », génère un fichier DLL et met à jour la bibliothèque de travail *workfile.*ini.

vamspice.exe –dir workfile.ini ↵
VamSpice affiche le contenu de la bibliothèque de travail.

Le fichier *cfunc.c* contient le corps du modèle VHDL-AMS écrit en langage C et adapté au langage XSPICE. Voir tableau 2-1 et l'exemple du modèle de la résistance.

Le fichier *ifspec.c* contient les déclarations des nœuds, des paramètres (*GENERICs*) et la configuration de SPICE OPUS.

	XSPICE	Langage C	VHDL-AMS	Description
Instruction séquentielles	nom_signal = forme_onde;	nom_signal = forme_onde;	[étiquette :] nom_signal <= forme_onde;	Affectation de signal
	OUTPUT_DELAY(port)	private->conn[x]->port[y] ->delay	valeur | expression [**after** expression_temps] {,…}	Forme d'onde
	nom_variable = expression;	nom_variable = expression;	[étiquette :] nom_variable := expression;	Affectation de variable
	if (…) [else]	if (…) [else]	**if** (…) **then** … [**else** …] **end if**;	Affectation séquentielle conditionnelle de signal

	switch (…) {
	case … : ….
	[default :…]
	}
		switch (…) {
		case … : …
		[default :…]
		}
			case … **is**
			when … => …
			[**when others** => …]
			end case;
				Affectation séquentielle sélective de signal

	while (1) {
	…
	}
		while (1) {
		…
		}
			loop
			…
			end loop;
				Instruction de boucle

	while (…) {
	…
	}
		while (…) {
		…
		}
			while (…) **loop**
			…
			end loop;	

	for (…;…;…[++|--]) {
	…
	}
		for (…;…;…[++|--]) {
		…
		}
			for … **in** … [**to**|**downto**] …
			loop
			…
			end loop;	

	if (…) continue | break;	if (…) continue | break;	**next** | **exit** [… **when** …];	Instructions *next* et *exit*
	assert(…);	assert(…);	**assert** … [**report** …] [**severity** …]	Instructions d'assertion

	if (…) … return …;	if (…) … return …;	**break** … **when** …;	Une partie instruction *break*
Instruction simultanées	exp_simple = expression ;	exp_simple = expression ;	[étiquette :] exp_simple == expression;	Instruction simultanée simple
	if (…) …else…	if (…) … else…	**if** (…) **use** … **else** … **end use**;	Instruction simultanée conditionnelle

	switch (…) {
	case … : ….
	[default :…]
	}
		switch (…) {
		case … : …
		[default :…]
		}
			case … **use**
			when … => …
			[**when others** => …]
			end case;
				Instruction simultanée sélective

Sous-programmes	(sous)type nom_fun ([…])
	{
	{déclarations}
	{instructions séqu.}
	return … ;
	}
		(sous)type nom_fun ([…]) {
		{déclarations}
		{instructions séqu.}
		return … ;
		}
			function nom_fun [(…)]
			return (sous)type **is**
			{déclarations}
			begin
			{instruction séqu.}
			return … ;
			end [**function**] nom_fun;
				Déclaration de fonction

	void nom_pro ([…]) {
	{déclarations}
	{instructions séqu.}
	}
		void nom_pro ([…]) {
		{déclarations}
		{instructions séqu.}
		}
			procedure nom_pro [(…)] **is**
			{déclarations}
			begin
			{instruction séqu.}
			end [**procedure**] nom_fun;
				Déclaration de procédure

Données du circuit	void <nom_modèle>(…)	void <nom_modèle>(…)	Nom de l'entité et de l'architecture	le modèle VHDL-AMS est représenté par une fonction
	ARGS	Mif_Private_t *private	Environnement du circuit	Argument standard pour tous les modèles (voir annexe)
	INIT	private→circuit.init	Initialisation des quantités	Initialisation de variable de simulation
	ANALYSIS = {DC, TRANSIENT, AC}	private→circuit.anal_type = {DC, TRANSIENT, AC}	DOMAIN = {QUIESCENT_DOMAIN, TIME_DOMAIN, FREQUENCY_DOMAIN}	Domaine d'analyse
	TIME	private→circuit.time	NOW	Le temps courant de simulation

	RAD_FREQ	private→circuit. frequency	FREQUENCY	La fréquence courante de simulation
Les paramètres	PARAM(param)	private→param[x]→element[y].rvalue	Paramètres du générique (un seul élément ou un vecteur)	Paramètre du circuit ou paramètre du modèle
	PARAM_SIZE(param)	private→param[x]→element[y].size	Nombre d'élément d'un paramètre du générique si ce lui ci est un vecteur	
	PARAM_NULL(param)	private→param[x].is_null	Utilise les paramètres par défaut	Si un des paramètres n'est pas spécifié dans le *.model* il prend la valeur par défaut
Les ports du circuit	INPUT(pin)	private→conn[x]→port[y] →input.rvalue	Les éléments du port pour les terminaux et les quantités de port	Les ports du modèle
	OUTPUT(pin)	private→conn[x]→port[y] →output.rvalue		
	INPUT_STATE(pin)	((Digital_t*)(private→conn[x] →port[y]→input.pvalue))→state	Les éléments du port pour les signaux	
	OUTPUT_STATE(pin)	((Digital_t*)(private→conn[x] →port[y]→output.pvalue))→state		
	PORT_SIZE(pin)	private→param[x].size	Nombre d'élément d'un port si ce lui ci est un vecteur	

Tableau 2-1 : Tableau de conversion de XSPICE vers VHDL-AMS

Exemple de conversion de VHDL-AMS vers SPICE d'une résistance : fichier resistance.vhd

```
-- Modèle VHDL-AMS d'une résistance
ENTITY resistance IS
 GENERIC (r :real:= 0.0);
 PORT (TERMINAL RIN, ROUT: Electrical);
END resistance;
ARCHITECTURE resistance_BODY OF resistance IS
 QUANTITY vr across ir through RIN TO ROUT;
BEGIN
 ir == vr / r;
END resistance_BODY;
```

La compilation de ce modèle à l'aide de l'instruction:
vamspice.exe –b resistance.vhd
génère les deux fichiers *cfunc.c* et *ifspec.c*

Le fichier *cfunc.c* est constitué de la façon suivante:

```
/* Inclusion des bibliothèques standard */
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

/* Inclusion des bibliothèques de XSPICE */
#include "cm.h"

/* Les paramètres ou les génériques */
double r = 0.0;

/* Les quantités libres et liées */
double vr;
double ir;

/* Les fonctions et les procédures de VHDL-AMS */

/* Le corps de l'architecture */
void cm_resistance(Mif_Private_t *private)
{
    Mif_Complex_t ac_gain;

    /* Initialisation des quantités libres et liée */
    if (private->circuit.init == 1) {
      vr = 0.0 ;
      ir = 0.0;
    }

    /* Recherche des valeurs des paramètres */
    r = private->param[0]->element[0].rvalue;

    /* Recherche des valeurs des quantités */
    vr = private->conn[0]->port[0]->input.rvalue - private->conn[1]->port[0]->input.rvalue ;

    /* Instructions simultanées simples */
    ir = vr / r;

    if (private->circuit.anal_type != MIF_AC ) {
      /* Les analyses DC et TRAN */
```

```
        private->conn[0]->port[0]->output.rvalue = ir ;
        private->conn[1]->port[0]->output.rvalue = -ir ;

        /* Recherche des dérivés partielles utilisées par SPICE */
        cm_analog_auto_partial();
      } else {
        /* L'analyse AC n'est pas utilisée */
        ac_gain.real = 0.0;
        ac_gain.imag = 0.0;
        private->conn[0]->port[0]->ac_gain[0].port[0] = ac_gain;
      }
}
```

Le fichier *ifspec.c* est constitué de la façon suivante :

```
/* Inclusions des bibliothèques standard */
#include <stdio.h>
/* Inclue les bibliothèques de XSPICE */
#include "spice.h"
#include "devdefs.h"
#include "ifsim.h"
#include "mifdefs.h"
#include "mifproto.h"
#include "mifparse.h"

/* Déclaration de la table des paramètres */
static IFparm MIFmPTable[] = {
      IOP( "r",             /* Nom du paramètre*/
             0,             /* Identificateur du paramètre */
          IF_REAL,          /* Type donnée du paramètre*/
            "r"             /* Description */
           ),
};
/* Type de port relatif à RIN*/
static Mif_Port_Type_t MIFportEnum0[] = {
       MIF_CONDUCTANCE,
       MIF_DIFF_CONDUCTANCE,
};
/* Type de port relatif à ROUT*/
static Mif_Port_Type_t MIFportEnum1[] = {
       MIF_CONDUCTANCE,
       MIF_DIFF_CONDUCTANCE,
};
/* Type de port en chaîne de caractère relatif à RIN :
  {"v","vd","i", "id","g","gd", "h", "hd","d" } */
static char *MIFportStr0[] = {
       "g",
       "gd",
};
/* Type de port en chaînes de caractère relatif à ROUT :
  {"v","vd","i", "id","g","gd", "h", "hd","d" } */
static char *MIFportStr1[] = {
       "g",
       "gd",
};
/* Le Tableau de connexion relatif à RIN et à ROUT */
static Mif_Conn_Info_t MIFconnTable[] = {
  {
     "RIN",                         /* Nom de la connexion */
```

```
    "RIN",                      /* Description de la connexion */
    MIF_INOUT,                  /* Type de la connexion : in, out ou inout */
    MIF_CONDUCTANCE,            /* Type de port par défaut */
    "g",                        /* Type de port par défaut en chaîne de caractères */
    2,                          /* Taille de type de port permis */
    MIFportEnum0,               /* Vecteur de type de port */
    MIFportStr0,                /* Vecteur de type de port en chaîne de caractères */
    MIF_FALSE,                  /* TRUE : si la connexion est un vecteur */
    MIF_FALSE,                  /* TRUE : s'il y a un tableau pour la limite supérieure */
    0,                          /* La taille du tableau pour la limite supérieure */
    MIF_FALSE,                  /* TRUE : s'il y a un tableau pour la limite inférieure */
    0,                          /* La taille du tableau pour la limite inférieure */
    MIF_FALSE,                  /* TRUE : si nul est lié à cette connexion */
 },
 {
    "ROUT",
    "ROUT",
    MIF_INOUT,
    MIF_CONDUCTANCE,
    "g",
    2,
    MIFportEnum1,
    MIFportStr1,
    MIF_FALSE,
    MIF_FALSE,
    0,
    MIF_FALSE,
    0,
    MIF_FALSE,
 },
};
/* Table des paramètres */
static Mif_Param_Info_t MIFparamTable[] = {
{
     "r",                                   /* Nom du paramètre */
     "r",                                   /* Description du paramètre */
     MIF_REAL,                              /* Type de paramètre : real, boolean, string, ... */
     MIF_TRUE,                              /* TRUE : s'il y a un valeur par défaut */
     {MIF_FALSE, 0, 0.0, {0.0, 0.0}, NULL},/*Valeur par défaut:boolean, integer, real, complex, string */
     MIF_TRUE,                              /* TRUE : s'il y a une limite inférieure */
     {MIF_FALSE, 0, 0.0, {0.0, 0.0}, NULL}, /* Valeur par défaut de la limite inférieure */
     MIF_FALSE,                             /* TRUE : s'il y a une limite supérieure */
     {MIF_FALSE, 0, 0.0, {0.0, 0.0}, NULL}, /* Valeur par défaut de la limite supérieure */
     MIF_FALSE,                             /* TRUE : si le paramètre est un vecteur */
     MIF_FALSE,                             /* TRUE : si le paramètre est associé à une connexion*/
     0,                                     /* L'indice du connecteur associé */
     MIF_FALSE,                             /* TRUE : s'il y a un tableau pour la limite supérieure */
     0,                                     /* La taille du tableau pour la limite supérieure */
     MIF_FALSE,                             /* TRUE : s'il y a un tableau pour la limite inférieure */
     0,                                     /* La taille du tableau pour la limite inférieure */
     MIF_TRUE,                              /* TRUE : si nul est lié à ce paramètre */
},
};

extern void cm_resistance(Mif_Private_t *);    /* L'entête du corps du modèle de la résistance */
static int val_terms                = 0;       /* Nombre de noms de terminaux */
static int val_numNames             = 0;       /* val_terms = val_numNames */
static int val_numInstanceParms     = 0;       /* Nombre de paramètres pour l'instance */
static int val_numModelParms        = 1;       /* Nombre de paramètres pour le .MODEL */
static int val_sizeofMIFinstance    = sizeof(MIFinstance);  /* Taille de structure de l'instance */
```

```
static int val_sizeofMIFmodel      = sizeof(MIFmodel);     /* Taille de structure du modèle */

/* Structure de manipulation du modèle ou composant */
SPICEdev cm_resistance_info      = {
{
      "resistance",                  /* Nom de ce type de modèle/composant */
      "resistance",                  /* Description de ce type de modèle/composant */
      &val_terms,                    /* Nombre de terminaux dans ce modèle */
      &val_numNames,                 /* Nombre de noms de terminaux dans termNames */
      NULL,                          /* Structure de nom de terminaux : termNames */
      &val_numInstanceParms,         /* Nombre de paramètre de l'instance */
      NULL,                          /* Table de noms de paramètre de l'instance */
      &val_numModelParms,            /* Nombre de paramètres du modèle */
      MIFmPTable,                    /* Table de noms de paramètres du modèle */
      cm_resistance,                 /* Pointe vers la fonction du corps du modèle */
      2,                             /* Nombre de connexions : nombre de ports */
      MIFconnTable,                  /* Table des informations des pins de connexion */
      1,                             /* Nombre de paramètres du modèle */
      MIFparamTable,                 /* Table des informations des paramètres du modèle */
      0,                             /* Nombre de paramètres de l'instance */
      NULL,                          /* Table des informations des paramètres de l'instance */
},
NULL,                                /* La routine pour entrer un paramètre dans une instance */
MIFmParam,                           /* La routine pour entrer un paramètre dans un modèle */
MIFload,                             /* La routine pour charger l'instance dans la matrice */
MIFsetup,      /* La routine d'organisation de l'instance avant de commencer à chercher la solution
*/
MIFunsetup,                          /* Effacer la mémoire avant de recommencer la simulation */
NULL,                                /* La routine d'organisation pour traiter l'analyse de PZ */
NULL,          /* La routine pour faire le traitement d'organisation dépendance en température */
MIFtrunc,                            /* La routine pour calculer d'erreur de troncature */
NULL,                                /* la routine pour chercher les équations de branche */
MIFload,       /* La routine pour charger l'instance dans la matrice pour l'analyse AC */
NULL,                                /* La routine pour chercher si un point est accepté ou non */
MIFdestroy,                          /* La routine pour détruire tous les modèles et les instances */
MIFmDelete,                          /* La routine pour détruire le modèle et toutes les instances */
MIFdelete,                           /* La routine pour détruire l'instance */
NULL,                                /* La routine pour les conditions initiales */
MIFask,                              /* La routine pour se renseigner sur les détails des paramètres */
MIFmAsk,       /* La routine pour se renseigner sur les détails des paramètres du modèle*/
NULL,          /* La routine pour charger la matrice pour l'analyse « pole-zero » */
MIFconvTest,                         /* La fonction test de convergence */
NULL,                                /* Routine pour organiser les informations de sensibilité */
NULL,                                /* Routine pour charger les informations de sensibilité */
NULL,                                /* Routine pour modifier les informations de sensibilité */
NULL,          /* Routine pour charger les informations de sensibilité pour l'analyse */
NULL,                                /* Routine pour afficher les informations de sensibilité */
NULL,                /* Routine pour calculer les erreurs pour les informations de sensibilité */
NULL,                                /* Routine pour l'opération de distorsion */
NULL,                                /* Routine pour l'analyse en bruit « noise » */
&val_sizeofMIFinstance,              /* Taille de structure de l'instance */
&val_sizeofMIFmodel,                 /* Taille de structure du modèle */
};
```

2.3 *VamSpiceEditor*

VamSpiceEditor est un éditeur de modèle VHDL-AMS et de *netlist* SPICE. Avec cet outil, nous pouvons compiler un modèle VHDL-AMS en appelant VamSpiceCompiler ou simuler une *netlist* SPICE en appelant SPICE OPUS. Cet outil, tient compte de la norme VHDL-AMS d'une

part et de la « norme » SPICE d'autre part, c'est à dire les mots clés sont collés en bleu et le commentaire en vert (*color key word*). Ceci facilite l'élaboration de modèle VHDL-AMS ou de *netlist* SPICE (Figure 2-7).

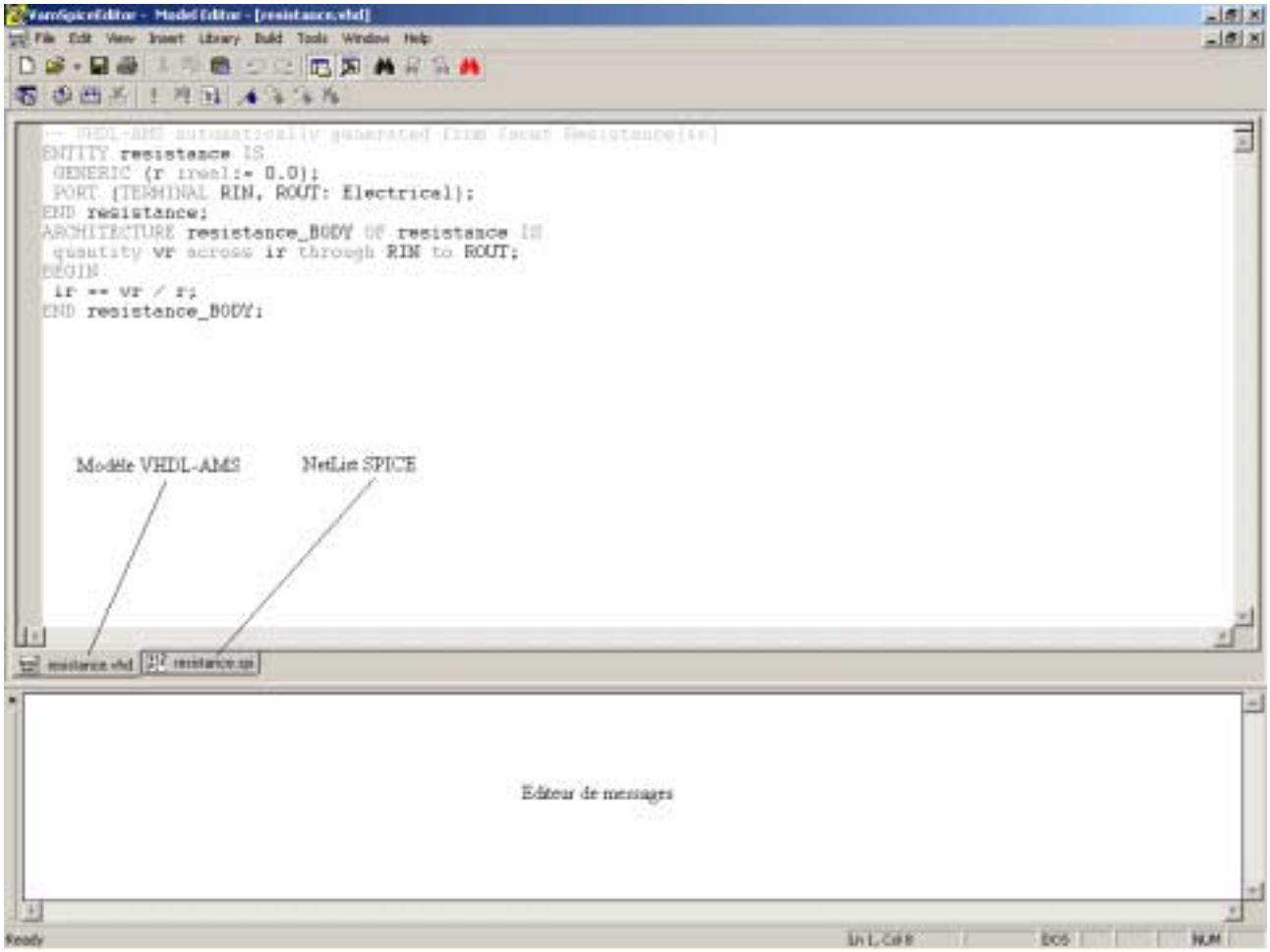


Figure 2-7 : VamSpiceEditor, Editeur de modèle VHDL-AMS et SPICE.

Cet outil sert de passerelle entre différent modules. Par exemple, il communique avec VamSpiceCompiler pour rediriger les résultats de compilation vers l'éditeur de messages de VamSpiceEditor. Il peut aussi communiquer en exécutant et mettant à jour les fichiers d'initialisation de SPICE OPUS. VamSpiceEditor peut aussi transférer et recevoir des informations de la schématique ELECTRIC. La paragraphe 2.6 traite en détail la communication entre les modules et le langage de communication.

Sur la facette VHDL-AMS, nous pouvons visualiser le contenu de la bibliothèque courante ou éliminer une entité et une architecture d'un modèle VHDL-AMS qui se trouve dans un fichier de travail (*workfile*). Sur la facette SPICE, nous pouvons commander la visionneuse de courbe (*Nutmeg*) pour changer les dimensionnements ou les couleurs d'affichage des courbes.

2.4 Le simulateur SPICE OPUS (= SPICE3 + XSPICE + Optimisation + *Nutmeg*)

SPICE OPUS [PUH98] est un simulateur électrique auquel a été ajouté un utilitaire d'optimisation. C'est une re-compilation du code source de Berkeley (SPICE3F5) original pour Windows 95/98/NT et Linux.

XSPICE [COX92] (*Georgia Tech Institute*) est un simulateur issu du programme de Berkeley auquel ont été ajoutés divers modules pour former un simulateur mixte. La caractéristique de XSPICE est de pouvoir accepter des modèles extérieurs appelés *code models* (CM). Les *code models* sont des modèles DLLs générés par VamSpice. La partie graphique (*Nutmeg*) du programme a été réécrite pour l'adapter à l'environnement Windows et Linux (Figure 2-8).

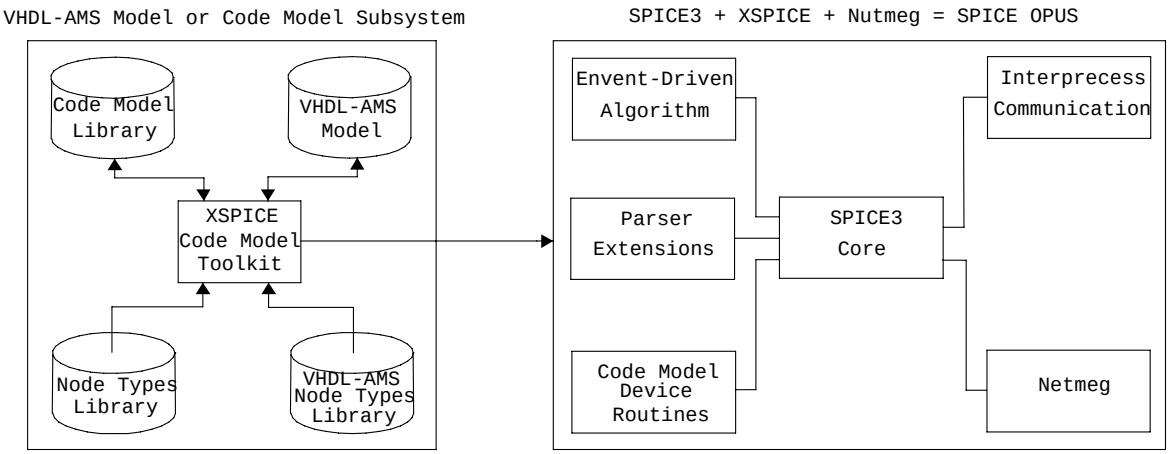


Figure 2-8 : Description d'un modèle VHDL-AMS, de SPICE OPUS, et la relation de XSPICE avec SPICE OPUS.

2.5 La schématique « *Electric*™ VLSI Design System »

Cette schématique est dédiée à la conception analogique et mixte VLSI. En ce qui concerne les circuits électriques, le principe est le même que n'importe quelle schématique CAO.

2.5.1 Icône, VHDL-AMS, circuit, et *netlist* SPICE

Dans une facette icône « {ic} », nous dessinons le schéma d'une icône qui correspond à un modèle VHDL-AMS. Sur la même vue, nous définissons les ports, les génériques et le modèle SPICE (*SPICE template*).

Les ports de l'instance sont typés (menu : *Export* → *Create Export*...), en ce sens qu'ils appartiennent à un domaine (une technologie) donné(e). Nous pouvons avoir dans une facette plusieurs ports appartenant à plusieurs domaines qui seront utilisés dans le modèle VHDL-AMS généré (figure 2-9 et figure 2-10, R_IN et R_OUT sont des ports de type *Electrical*).

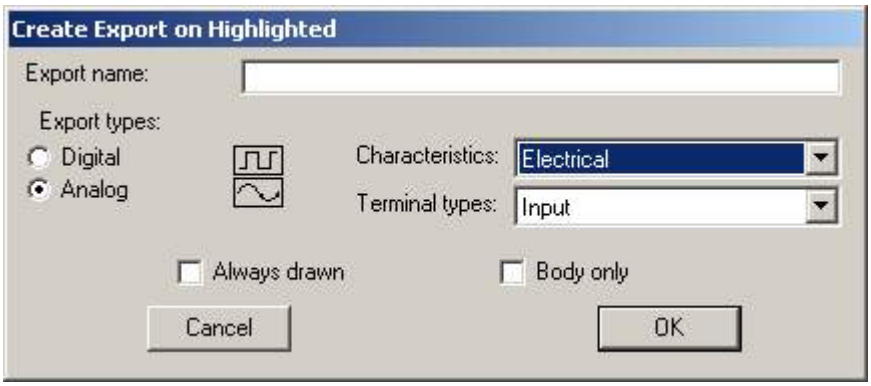


Figure 2-9 : Création des ports mixtes ou dans un domaine donné (Electrical, Thermal, …).

Chaque icône ou chaque modèle VHDL-AMS doit communiquer avec le monde extérieur par le biais des paramètres (GENERICs). Dans le cas de la définition d'une instance dans une facette icône, nous pouvons ajouter des paramètres (menu : *Tools→VHDL-AMS Compiler→ Add VHDL-AMS Generic*) qui seront utilisés par la suite dans le modèle VHDL-AMS généré et dans la facette test de cette icône (Figure 2-10).

Les paramètre sont définit par :

<param_name>=[<actual_value>] | [(?|<actual_value>);def=<default_value>]

avec :

<param_name>	: nom du paramètre à définir.
<actual_value>	: la valeur actuelle à définir.
? | <actual_value>	: la valeur actuelle n'est pas définie (?) ou elle est définie (actual_valeur).
def=<default_value>	: «*default_value*» est la valeur par défaut.

L'instruction suivante est définie dans le SPICE *template* :

A$(node_name) $(node1)…$(nodeN) m$(node_name) \n .model m$(node_name) <facet_name> [( ] param1=$(param1)…paramN=$(paramN) [ )]

Avec :

node_name	: nom de l'icône dans une facette schématique « {sch} ».
$(xxxxx)	: une variable (un paramètre) qui est remplacé par une chaîne de caractères (par un valeur) à la génération de la *netlist* SPICE.

SPICE template (menu : *Tools → Simulation (SPICE) → Set SPICE Template*) se compose de deux parties :

- La première partie fait appel au composant par l'intermédiaire de la lettre **A** et le nom de l'instance, ensuite sont définits les ports du composant (**$(node1)…$(nodeN)**) et le nom du modèle (**m$(node_name)**),
- La deuxième partie a pour but de définir le modèle du composant, à travers lequel, on peut changer les valeurs de paramètres (les génériques).

Ces deux parties sont séparées par un retour chariot « **\n** ». Sur la facette de la figure 2-10, nous modélisons schématiquement une résistance au niveau comportemental.

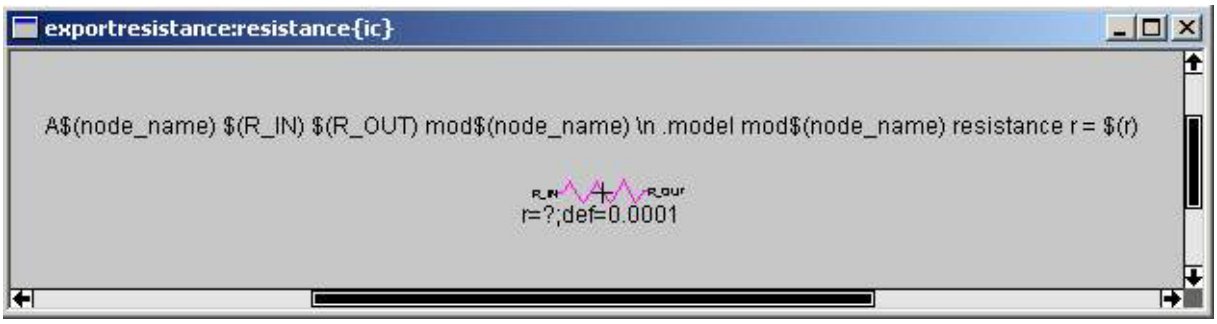


Figure 2-10 : Icône d'une résistance qui correspond à un modèle VHDL-AMS.

Une fois que toutes ces parties sont définies, nous faisons générer automatiquement (menu : *View→ Make VHDL-AMS View*) le corps d'un modèle VHDL-AMS, c'est à dire, l'entité, les génériques, les ports, et une architecture vide. Ensuite, nous ajoutons les éléments de description du modèle relatifs à un niveau d'abstraction donné (Figure 2-11).

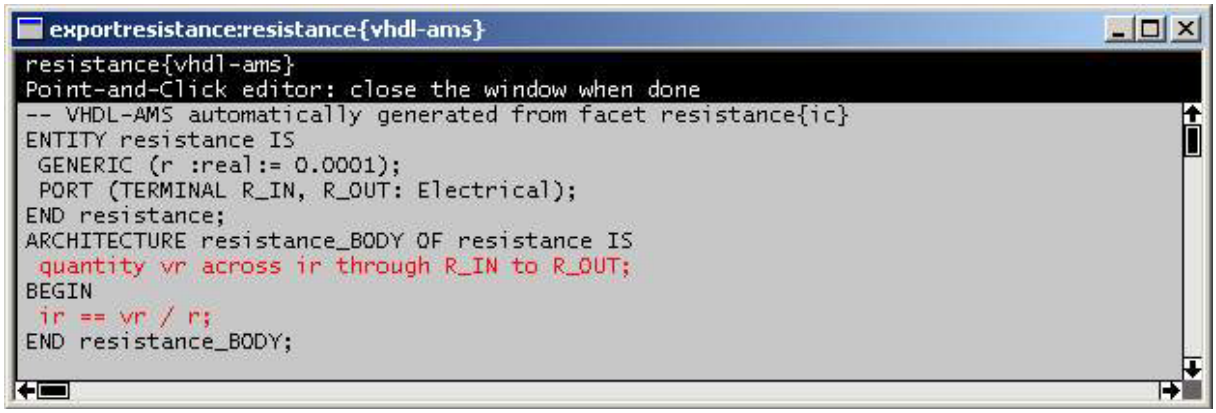


Figure 2-11 : Le modèle VHDL-AMS généré automatiquement à partir de l'icône figure 2-10.

Dans une vue schématique « {sch} », nous dessinons le schéma du circuit test de notre composant (Figure 2-12), en appelant l'icône qui est définie en haut. Ensuite, nous ajoutons les entrées, le type d'analyse et les courbes à afficher et nous lançons la simulation. Nous générons automatiquement une « *netlist* » analogique utilisée par SPICE OPUS pour simuler.

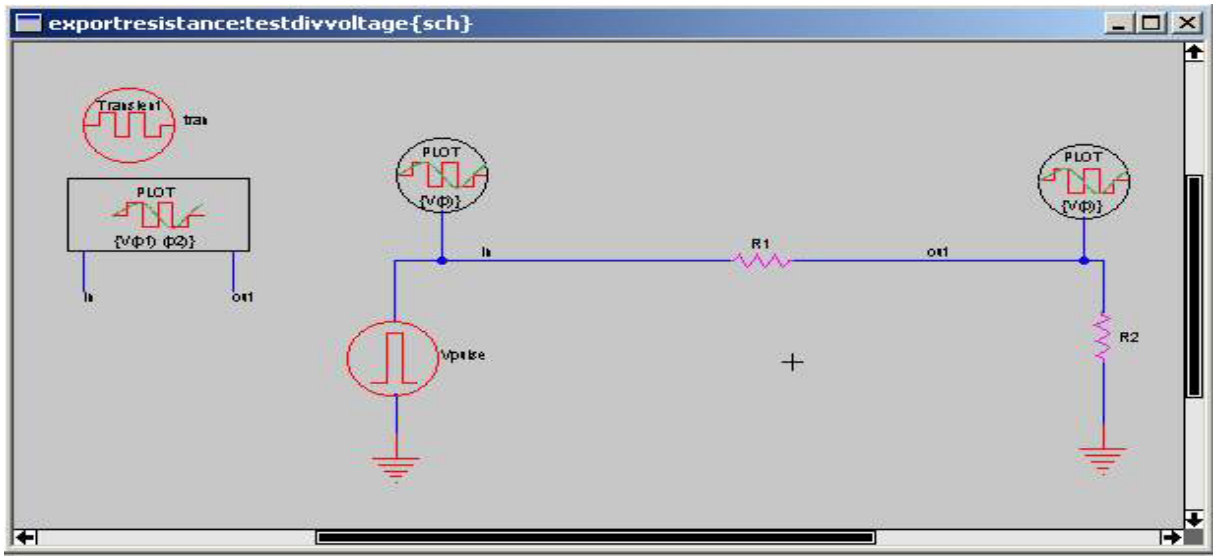


Figure 2-12 : Vue schématique {sch} pour tester le modèle VHDL-AMS.

2.5.2 Macro-composant, icône et hiérarchie

Des macro-composants peuvent être obtenus en utilisant la procédure précédente et transformés en icônes. Une icône est un rectangle dans lequel on ne peut pas visualiser un sous-circuit, mais seulement le nom du macro-composant (Figure 2-13). La propriété importante des icônes est de pouvoir apparaître en tant que composants eux-mêmes dans des circuits grâce à des commandes de hiérarchie montante ou descendante (Figure 2-14).

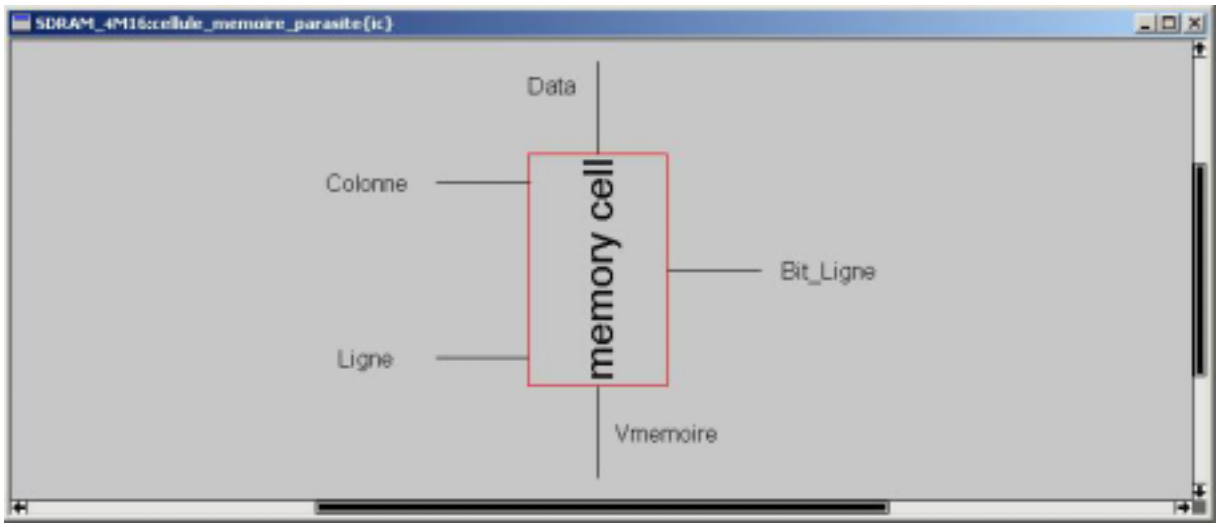


Figure 2-13 : Une icône qui représente un sous-circuit.

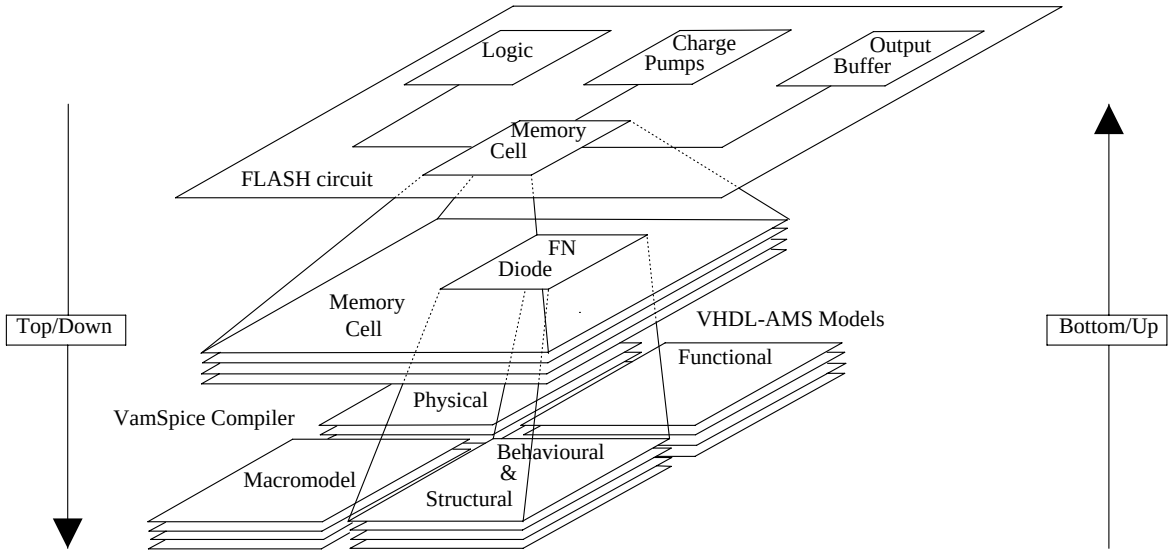


Figure 2-14 : Exemple de l'approche de la méthodologie schématique.

2.5.3 Une bibliothèque VHDL-AMS (*VHDL-AMS Libraries Parts*)

Pour avoir une bonne méthodologie de conception, il faut avoir un support logiciel qui organise les étapes et les niveaux de conception. Nous n'avons pas besoin d'aller chercher une

partie de conception dans un fichier dans le cas où ce lui ci se compose de plusieurs fichiers. Ce support logiciel sera aussi toujours en permanence associé à l'outil Electric™.

Nous pouvons comparer cette idée aux tiroirs d'une armoire (Tableau 2-2) où chaque tiroir est désigné par un nom. Chaque tiroir se compose de plusieurs compartiments qui porte également un nom. Nous pouvons arranger nos composants dans ces compartiments. L'idée de ce support provient de l'agencement en tiroirs modélisable par VHDL-AMS à différents types de niveaux : multi-niveau (multi-abstraction [HER02]) et le multi-domaine (multi-technologique). Nous y ajoutons la multi-organisation (multi-arrangement). La figure 2-15, nous montre un exemple d'organisation de la bibliothèque.

Armoire	Multi-domaine
Tiroir	Multi-niveau
Compartiment	Multi-organisation

Tableau 2-2 : Une équivalence d'organisation.

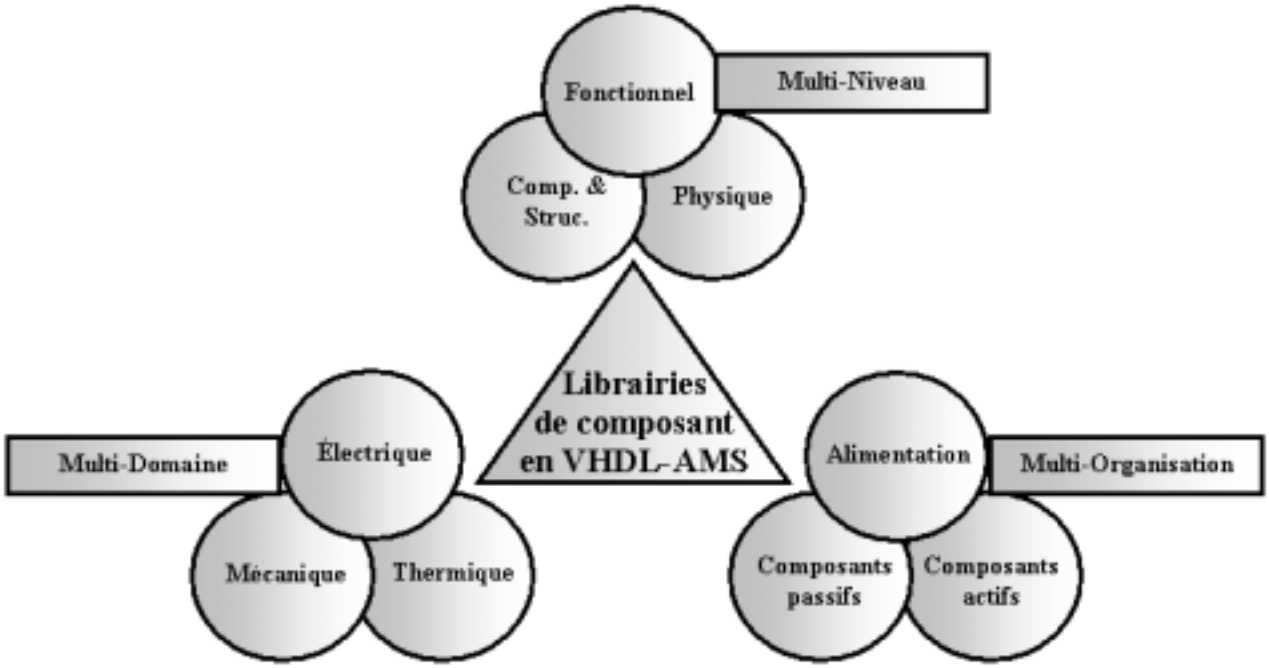


Figure 2-15 : Une façon d'organiser la bibliothèque de composant en VHDL-AMS.

2.5.3.1 Organisation de la bibliothèque

Comme il est motionné plus haut, la bibliothèque se structure en 3 niveaux d'organisation (Figure 2-15) du plus général vers le plus spécifique. Le niveau 1 est un niveau d'organisation générale, c'est à dire d'appartenance à un certain groupe, certaine famille, ou certain niveau d'abstraction. Ce niveau ne reçoit pas de composants (voir *State: Never used*). Le niveau 2 est un niveau d'organisation qui peut aussi recevoir un composant (voir *State: Not Used* ou *Used*). Le niveau 3 ne peut recevoir que des composants (voir *State: Not Used* ou *Used*) (Figure 2-16).

Figure 2-16 : Exploration de la bibliothèque.

2.5.3.2 Transport du composant

Une fois que le composant est défini par la procédure décrite plus haut. Nous testerons le composant (compilation, simulation, etc.) avant de le transporter.

Le transport (menu : *Tools→ VHDL-AMS Labrary Parts→ Export Part*) du composant vers la bibliothèque se compose en 6 phases :

- Phase 1 : c'est une phase de compilation de SPICE *template* et de la facette courante

```
Phase 1: Compile template and facet...
Compiling...
A$(node_name) $(R_IN) $(R_OUT) mod$(node_name) \n .model mod$(node_name) resistance r = $(r)

resistance{ic} - 0 error(s), 0 warning(s)
```

Le compilateur de SPICE *template* a été réalisé entièrement à la main c'est à dire que toutes les phases du compilateur sont programmés à la main. Ce compilateur est divisé en trois phases (Figure 2-17) :

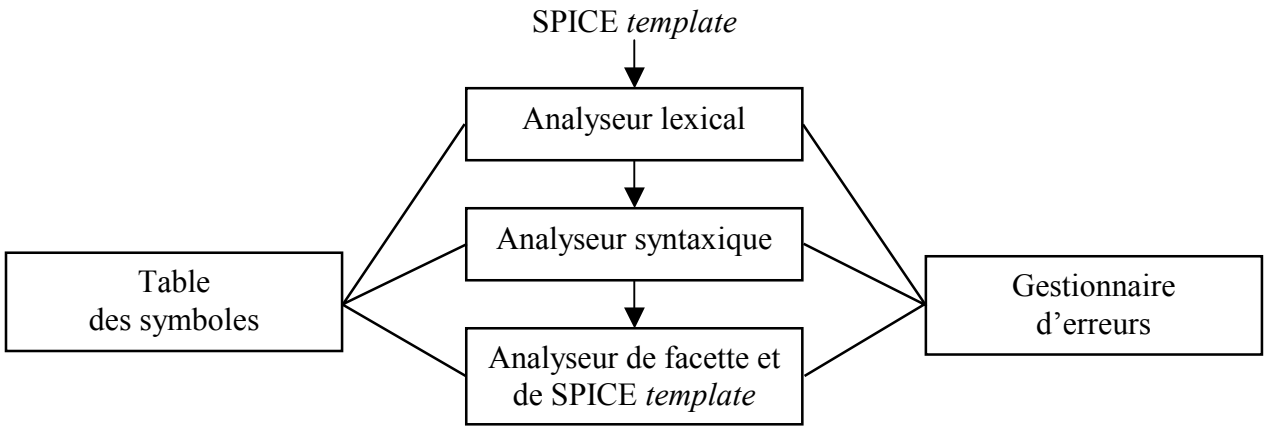


Figure 2-17 : Phases de compilation de SPICE *template*.

L'analyseur lexical lit les caractères formant le SPICE *template* et les groupe en un flot d'unités lexicales (***token***), dont chacune représente une suite de caractères formant un mot à analyser. Les mots analysés sont séparés par un espace.

La phase d'analyse syntaxique consiste à regrouper les unités lexicales du SPICE *template* en structures grammaticales qui seront employées par le compilateur pour comparer certains éléments de l'instruction de SPICE *template* avec la facette de l'icône.

A l'issue de l'analyse lexicale et de l'analyse syntaxique, la phase finale de notre compilateur est la vérification ou la comparaison de certains éléments de l'instruction de SPICE *template* avec la facette de l'icône à compiler. Par exemple, le compilateur compare les ports et l'ordre de l'écriture des ports par rapport aux ports qui sont définis au niveau de l'icône.

Le compilateur collecte les identificateurs utilisés dans SPICE *template*. Il enregistre toutes les informations dans la table de symboles. Cette dernière est divisée en deux sous-tables, la première appelée par la structure « AINSTANCE » et la deuxième par « CARDMODEL » (Figure 2-18 et Figure 2-19)

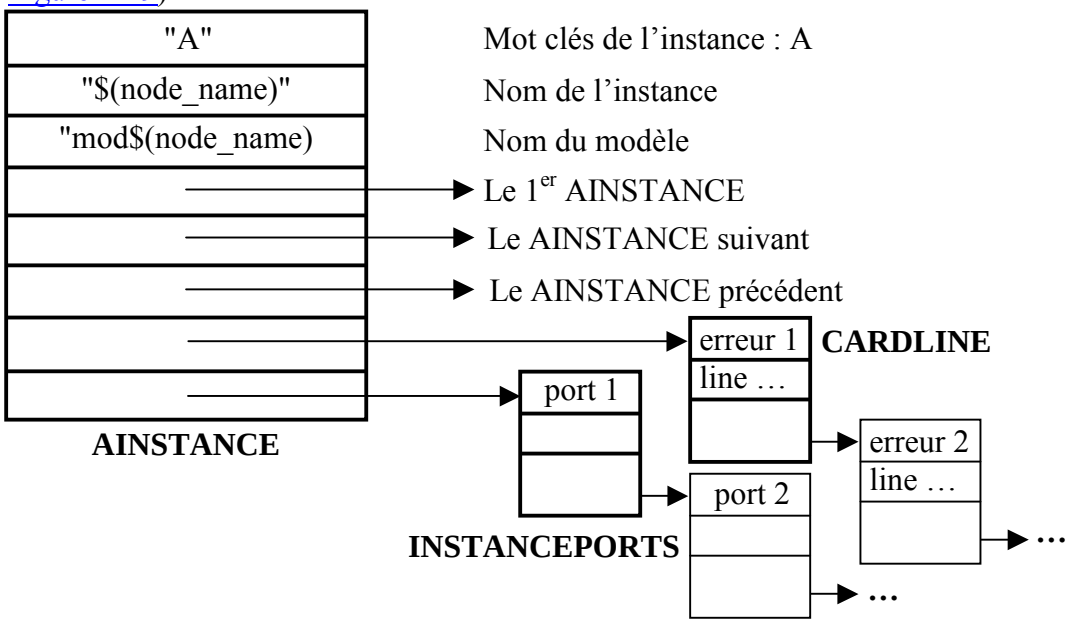


Figure 2-18 : La structure de AINSTANCE.

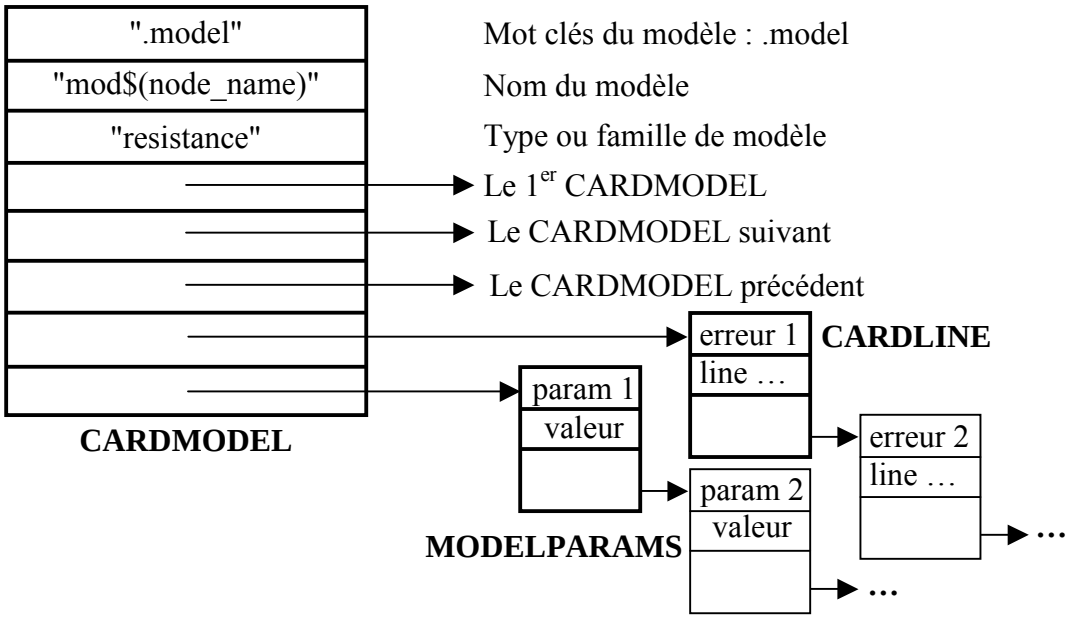


Figure 2-19 : La structure de CARDMODEL.

L'analyseur syntaxique et l'analyseur de facette et du SPICE *template* traitent une grande part des erreurs (Figure 2-18 et Figure 2-19). L'analyseur lexical peut détecter quelques erreurs surtout à la fin de l'instruction où les caractères restant à lire ne peuvent former aucune unité lexicale de l'instruction.

Exemples des erreurs générées par le compilateur :

- Au lieu de mettre 'A' nous mettons 'C' pour désigner une capacité au lieu d'une instance externe. Le compilateur regarde d'abord si l'instruction commence bien par 'A'. Le message d'erreur généré par l'analyseur lexical et l'analyseur syntaxique est le suivant.

```
Phase 1: Compile template and facet...
Compiling...
C$(node_name) $(R_IN) $(R_OUT) mod$(node_name) \n .model mod$(node_name) resistance r = $(r)
^ : Error (1) :
'C' : instance must begin by 'A' or 'a'

resistance{ic} - 1 error(s), 0 warning(s)
```

- Par erreur de frappe, nous avons par exemple oublié de mettre un point '**.**' devant '**model**'. Le compilateur détecte cette erreur car '*.model*' est un mot clé de l'instruction. Le message d'erreur générée par l'analyseur lexical est le suivant :

```
Phase 1: Compile template and facet...
Compiling...
A$(node_name) $(R_IN) $(R_OUT) mod$(node_name) \n model mod$(node_name) resistance r = $(r)
A$(node_name) $(R_IN) $(R_OUT) mod$(node_name) \n ^ : Error (51) :
'model' : model card must begin by '.MODEL' or '.model'

resistance{ic} - 1 error(s), 0 warning(s)
```

- Les deux ports dans l'instruction de SPICE *template* ont été permutés par erreur. Le compilateur compare les ports de facette avec SPICE *template* d'où les erreurs détectées à la phase de comparaison :

```
Phase 1: Compile template and facet...
Compiling...
A$(node_name) $(R_OUT) $(R_IN) mod$(node_name) \n .model mod$(node_name) resistance r = $(r)
A$(node_name) $(^ : Error (17) :
'R_OUT' : is different to 'R_IN' the port in this facet

A$(node_name) $(R_OUT) $(R_IN) mod$(node_name) \n .model mod$(node_name) resistance r = $(r)
A$(node_name) $(R_OUT) $(^ : Error (26) :
'R_IN' : is different to 'R_OUT' the port in this facet

resistance{ic} - 2 error(s), 0 warning(s)
```

  - Phase 2 : c'est la phase du choix de la bibliothèque où nous transportons le composant (Figure 2-20).

Figure 2-20 : Choix de la bibliothèque cible.

```
Phase 2: Choose library parts to export resistance{ic}...
      Current library parts : 'Analog'...
```

  - Phase 3 : c'est la phase de transport du composant et d'enregistrement de la bibliothèque cible. Si nous utilisons les champs de '*menu name*' et '*menu description*', c'est à dire que nous écrivons le nom et le description du compostant. Il va créer automatiquement un niveau 3. Si nous n'utilisons pas ces champs, il va utiliser le niveau 2, et dans ce cas, ce niveau n'est plus un niveau d'organisation (Figure 2-21).

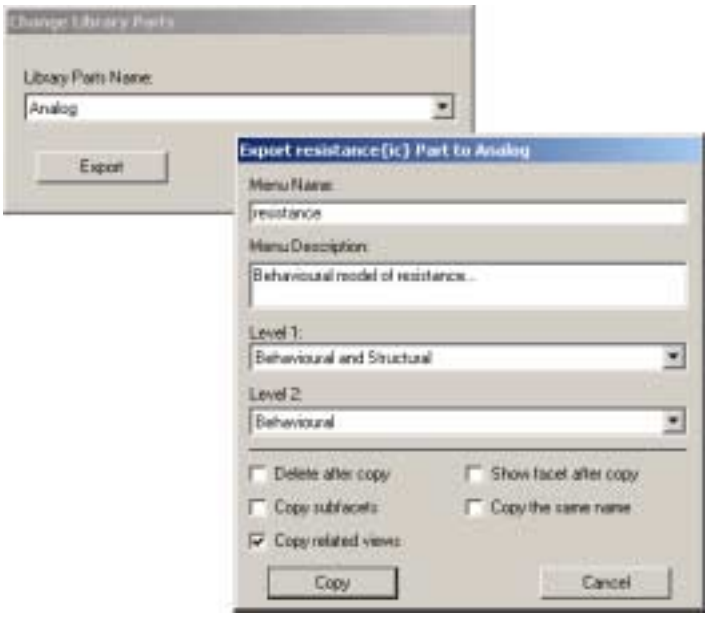


Figure 2-21 : Description et transport du composant.

```
Phase 3: Export resistance{ic} and Save library parts...
Copied ExportComponent:AMS_resistance_1036007246{ic} to library AMS_Analog_1035973828
last resistance{vhdl-ams}
Copied ExportComponent:AMS_resistance_1036007246{vhdl-ams} to library AMS_Analog_1035973828
C:\VamSpiceDesigner\VamSpiceSchematic\lib\vhdlams_plib\AMS_Analog_1035973828.elib written on
Thu Apr 17 16:06:09 2003 (6 facets)
```

- Phase 4 : c'est une phase de test de vérification du bon fonctionnement du transport.

```
Phase 4: 'AMS_resistance_1036007246{ic}' created in 'AMS_Analog_1035973828'
```

- Phase 5 : c'est une phase d'effacement des anciennes versions du composant.

```
Phase 5: Delete old facet versions...
No unused old facet versions to delete
```

- Phase 6 : c'est une phase de compilation du modèle VHDL-AMS transporté.

```
Phase 6: Compiling the VHDL-AMS model...
VamSpice Copyright (c) 2002 GET
Created by ENST / COMELEC Department
All rights reserved
Compiling...
AMS_resistance_1036007246.vhd
Linking...
Warning: no export definition file provided
dllwrap will create one, but may not be what you want
Update 'AMS_resistance_1036007246.vhd' model to the work file.
AMS_resistance_1036007246.cm - 0 error(s), 0 warning(s), highest severity 0
```

2.6 Lien entre les outils

Comme il est décrit plus haut, l'outil VamSpiceDesigner est composé de 5 modules (ou programmes). Chaque module fonctionne indépendamment les uns des autres (Figure 2-22).

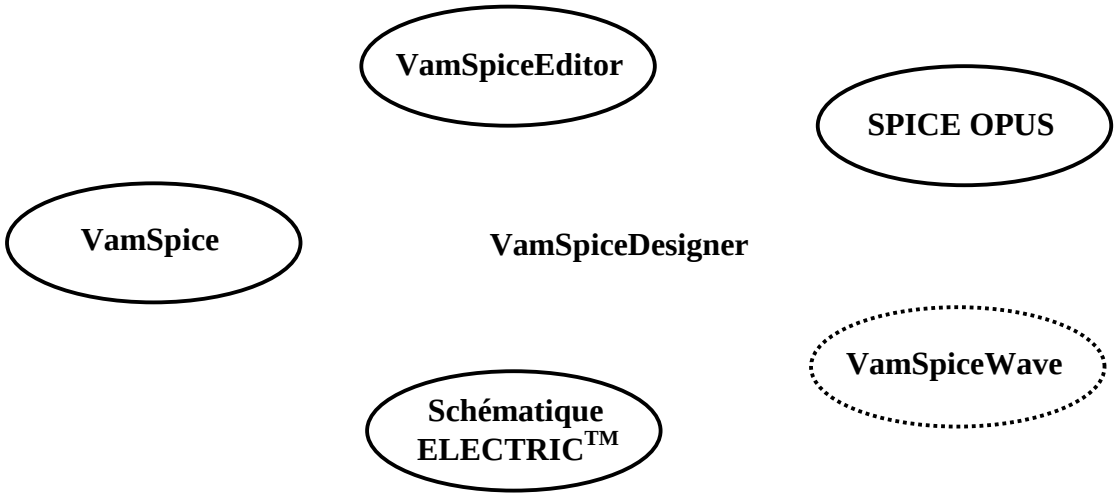


Figure 2-22 : Groupe des outils de VamSpiceDesigner.

Si nous voulons utiliser le schématique, nous faisons en sorte qu'elle devienne le maître et qu'elle communique avec les autres modules à travers VamSpiceEditor. Si nous ne voulons utiliser que l'éditeur de modèle celui-ci sera le maître (Figure 2-23).

Pour communiquer entre modules, nous avons mis en place un langage de communication. Ce langage concerne les modules suivants : VamSpiceEditor ↔ Electric, VamSpiceEditor ↔ VamSpiceWave, et Electric ↔ VamSpiceEditor ↔ VamSpiceWave.

<Destination> <Nom fichier> <Ordre d'avant exécution > <Ordre d'exécution> <Ordre d'après exécution > <Expéditeur>

<Destination>	: VAMSPICEEEDITOR | VAMSPICEWAVE | ELECTRIC
<Ordre d'avant exécution>	: QUIT_SPICEOPUS | QUIT_VAMSPICEEDITOR | QUIT_VAMSPICEWAVE | QUIT
QUIT_XXXXX	: Quitter après l'exécution d'une tache XXXXX.
QUIT	: L'application qui reçoit cet ordre reste actif
<Ordre d'exécution>	: COMPILE | BUILD | SIMULATE | WAVE | TRAME
COMPILE | BUILD	: Ordre pour activer le compilateur VamSpice
SIMULATE	: Ordre de simulation (SpiceOpus)
WAVE	: Ordre pour afficher les courbes (VamSpiceWave)
TRAME	: Ordre de transférer des données.
<Expéditeur>	: VAMSPICEEEDITOR | VAMSPICEWAVE | ELECTRIC avec : **<Destination>** != **<Expéditeur>** (voir exemple).

Exemple :

ELECTRIC envoie un ordre de compiler un modèle VHDL-AMS et quitter l'éditeur de modèle (compilateur) après exécution (Figure 2-23).

ELECTRIC :

❶ **VAMSPICEEDITOR d:\exemple\diode.vhd QUIT_SPICEOPUS BUILD QUIT_VAMSPICEEDITOR ELECTRIC**

VamSpiceEditor reçoit l'ordre, il charge le fichier à exécuter (diode.vhd), il met un terme à l'exécution de SpiceOpus pour qu'il libère le fichier du code modèle (cm/DLL), si ce lui-ci est en cours d'exécution, et il examine s'il y a un retour d'informations par les mots clé BUILD | COMPILE {| SIMULATE}.

VamSpiceEditor :

❷ **ELECTRIC d:\exemple\diode.vhd QUIT TRAME QUIT VAMSPICEEDITOR**

VamSpiceEditor commence à envoyer les trames des informations de compilation vers ELECTRIC et à la fin, il quitte l'application.

VamSpice est un outil qui fonctionne sur un environnement DOS et les autres applications fonctionnent sur un environnement Windows. Donc, le seul moyen pour récupérer les résultats de compilation de VamSpice, c'est de créer un fichier mémoire (ou un canal mémoire) pour rediriger les informations vers des applications Windows.

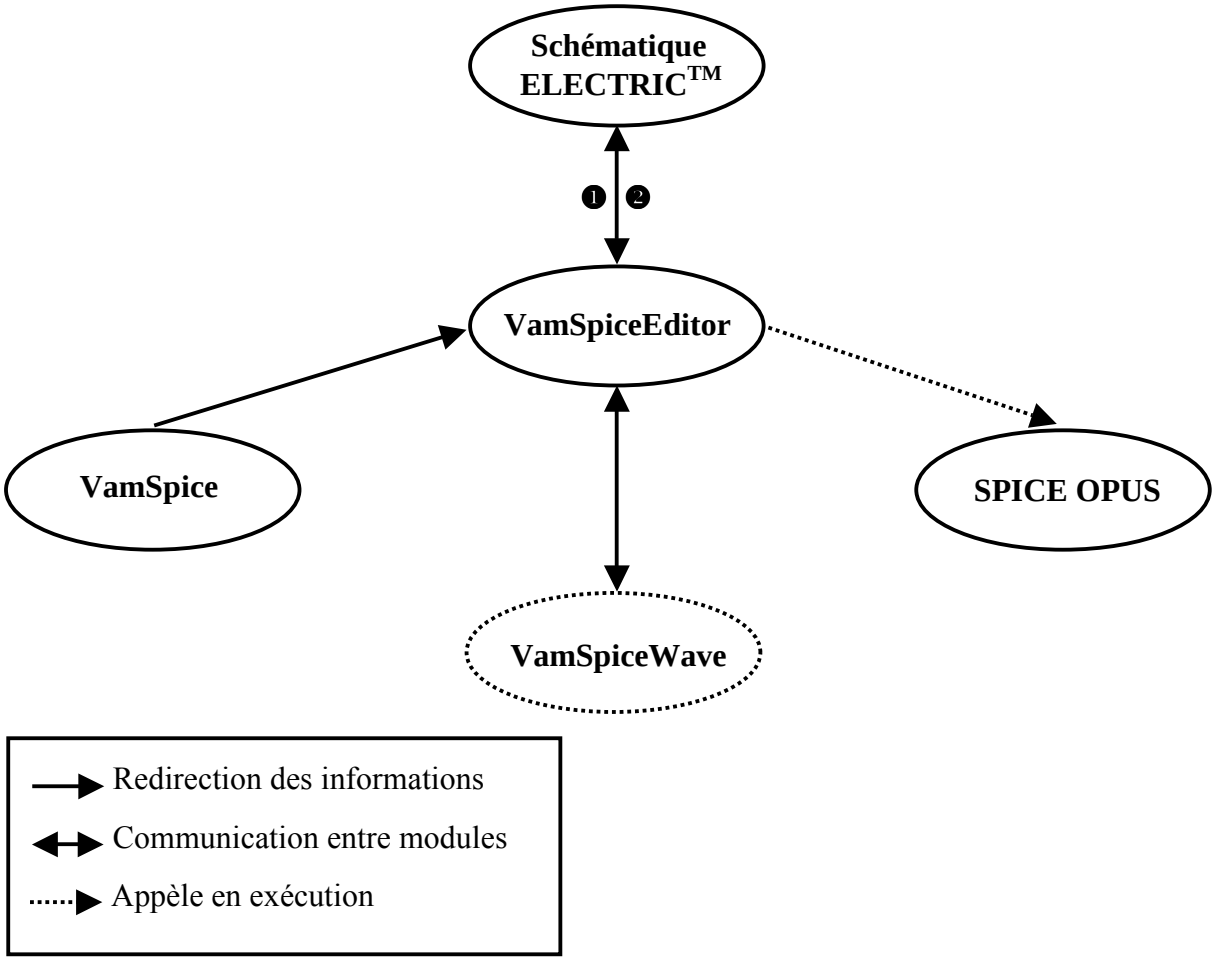


Figure 2-23 : Communication entre modules.

2.7 Exemples d'application

Pour les modèles VHDL-AMS, les icônes sont en violet sur l'écran, les modèles de XSPICE sont en bleu clair et les modèles de composant de type SPICE sont en rouge.

2.7.1 Modélisation fonctionnelle

2.7.1.1 Valeur absolue

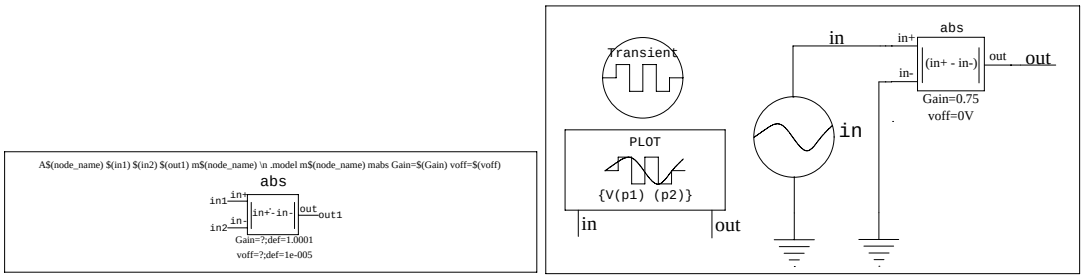


Figure 2-24 : Icône et schéma test d'un modèle VHDL-AMS de valeur absolue.

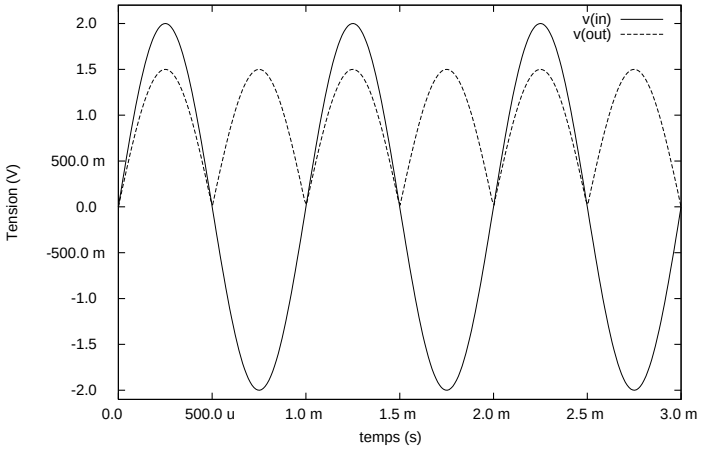


Figure 2-25 : Résultat de simulation d'un modèle VHDL-AMS de valeur absolue

2.7.1.2 Tangente hyperbolique

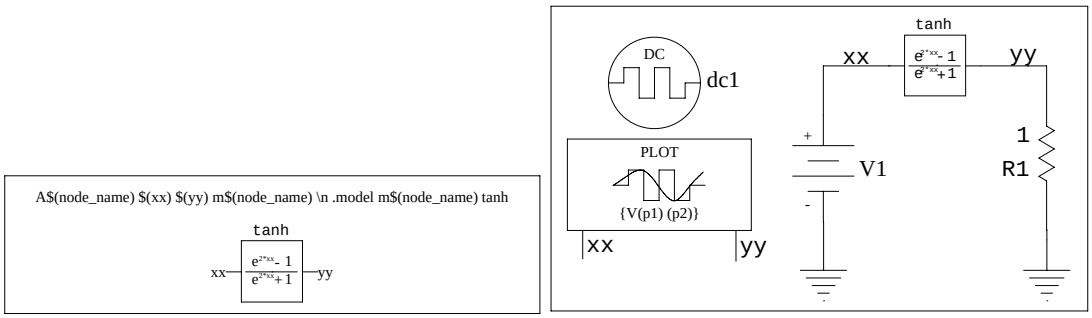


Figure 2-26 : Icône et schéma test d'un modèle VHDL-AMS de tangente hyperbolique

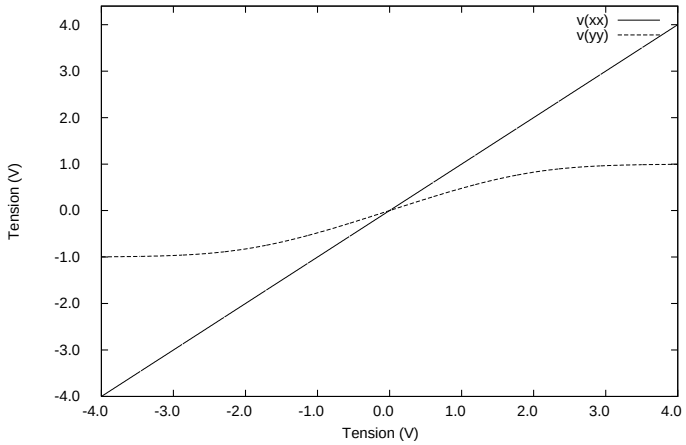


Figure 2-27 : Résultat de simulation d'un modèle VHDL-AMS de tangente hyperbolique

2.7.1.3 Différenciateur-multiplieur

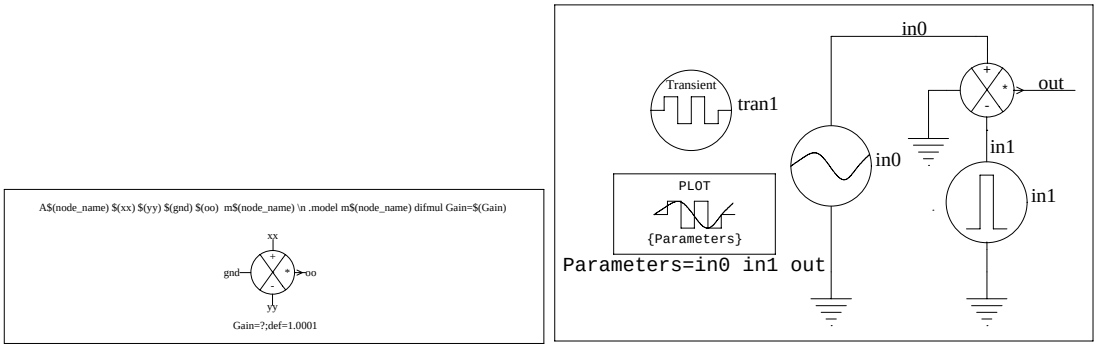


Figure 2-28 : Icône et schéma test d'un modèle VHDL-AMS de différenciateur-multiplieur

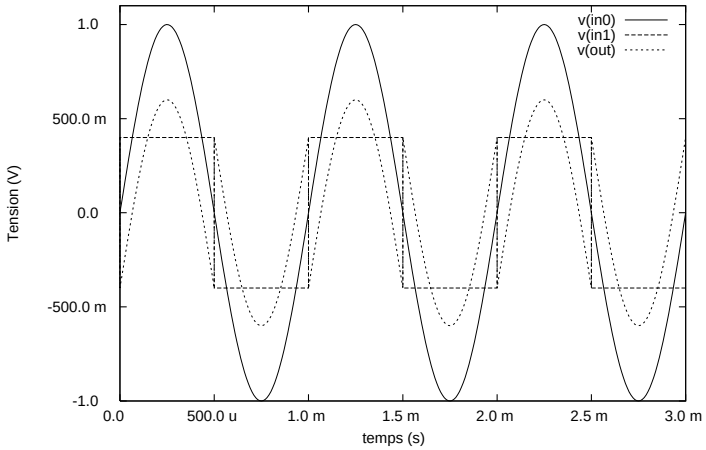


Figure 2-29 : Résultat de simulation d'un modèle VHDL-AMS de différenciateur-multiplieur.

2.7.1.4 Comparateur de tension

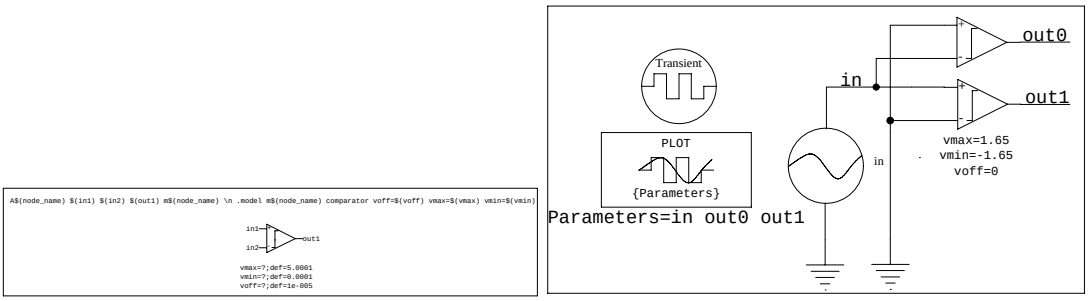


Figure 2-30 : Icône et schéma test d'un modèle VHDL-AMS de comparateur de tension

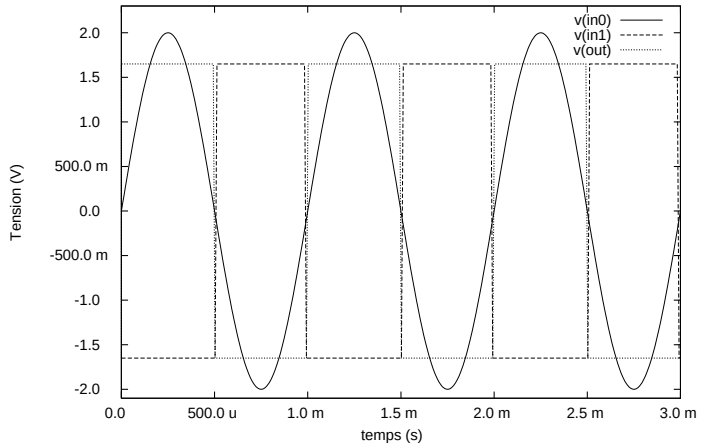


Figure 2-31 : Résultat de simulation d'un modèle de comparateur de tension.

2.7.1.5 Intégrateur-gain

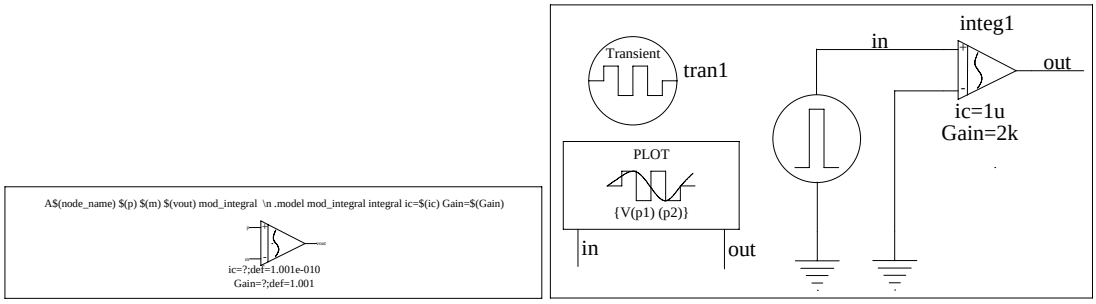


Figure 2-32 : Icône et schéma test d'un modèle VHDL-AMS d'intégrateur-gain.

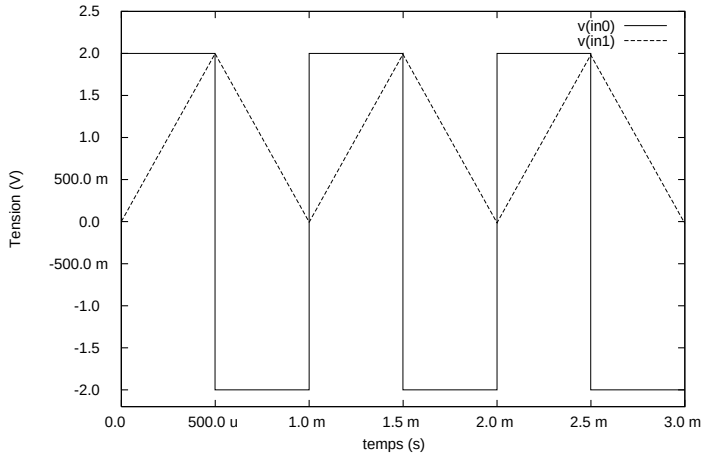


Figure 2-33 : Résultat de simulation d'un modèle d'intégrateur-gain.

2.7.2 Modélisation comportementale

2.7.2.1 Résistance

Cet exemple est décrit plus haut (Figure 2-10 à Figure 2-12). Pour le circuit test, nous prenons un simple diviseur de tension. Nous excitons ce circuit avec un générateur d'impulsion à l'entée et nous visualisons la variation du signal de sortie (Figure 2-34).

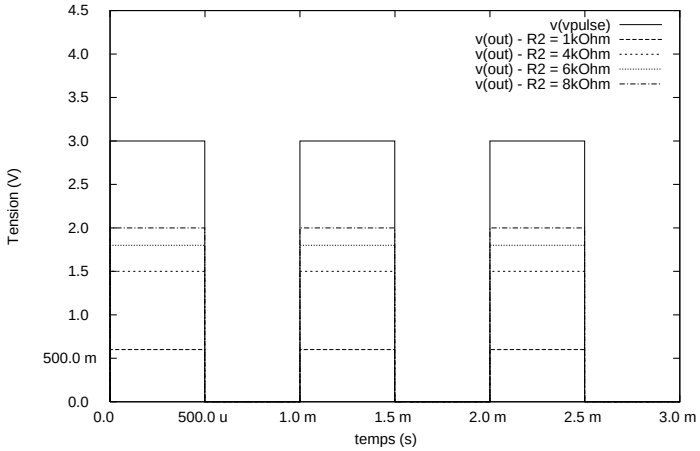


Figure 2-34 : Un simple diviseur de la tension (avec R2 variable et R1 = 4kΩ).

2.7.2.2 Capacité

Dans cet exemple, nous modélisons avec VamSpiceDesigner une capacité au niveau comportemental. Le figure 2-35-a, qui représente l'icône d'une capacité correspondant à un modèle VHDL-AMS avec ***c_in*** et ***c_out*** les ports de ce modèle et ***c*** comme paramètre générique. Pour tester ce modèle, nous prenons l'exemple d'un filtre passif à cellule RC (Figure 2-35-b) avec R une résistance de type SPICE. Nous excitons ce circuit par un générateur d'impulsion à l'entrée et nous regardons la sortie toute en faisant varier la capacité ***C1*** (Figure 2-36).

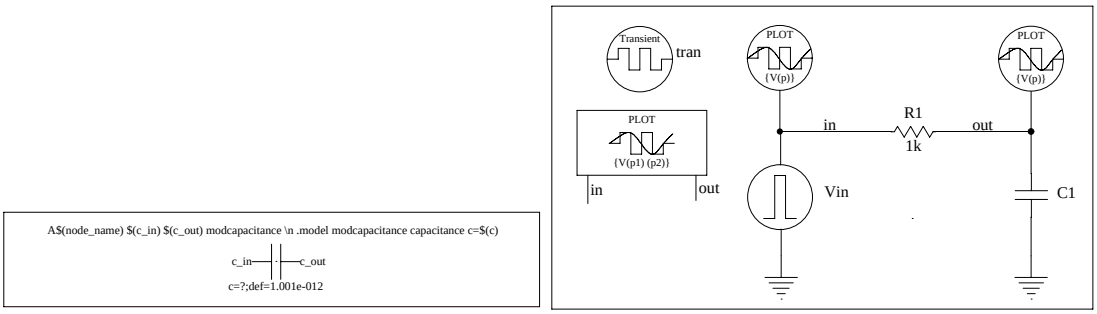


Figure 2-35 : (a) Icône et (b) circuit test d'une capacité qui correspondant à un modèle VHDL-AMS.

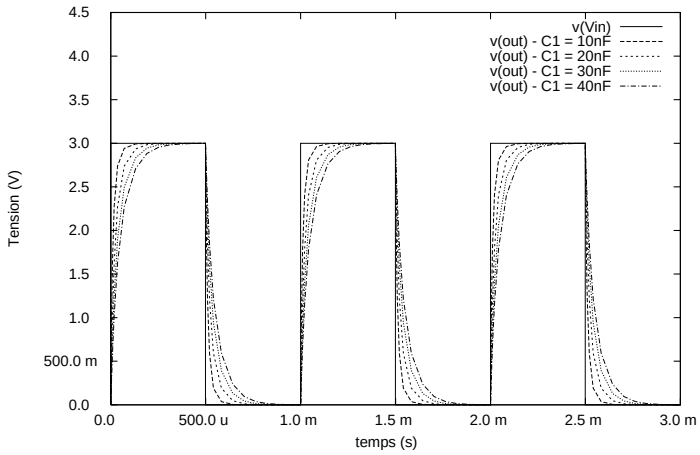


Figure 2-36 : Charge et décharge d'une capacité modélisée avec VHDL-AMS (avec C1 variable et R1 = 1kΩ).

2.7.2.3 Filtre passif 1er ordre

Cet exemple est le même que celui décrit plus haut, mais au lieu de prendre une résistance de type SPICE, nous prenons une résistance modélisée par VHDL-AMS. L'intérêt de cet exemple et l'exemple précédent est de pouvoir modéliser un circuit en mixant du SPICE avec du VHDL-AMS.

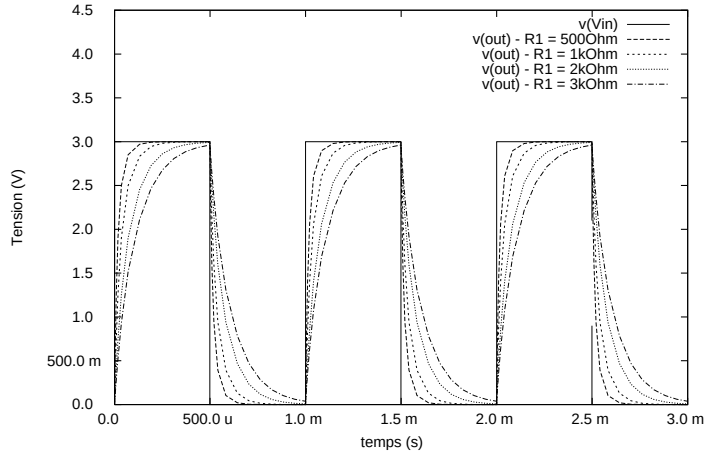


Figure 2-37 : Résistance et capacité modélisé avec VHDL-AMS (avec R1 variable et C1 = 30nF).

2.7.2.4 Diode

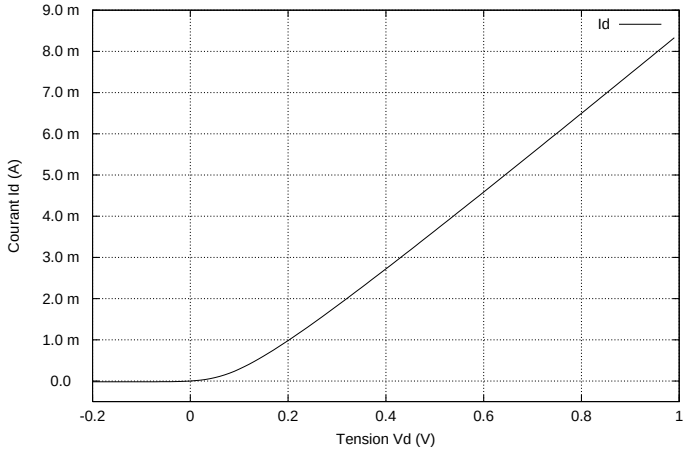


Figure 2-38 : Caractéristique d'une diode modélisée avec VHDL-AMS (Température 27°C).

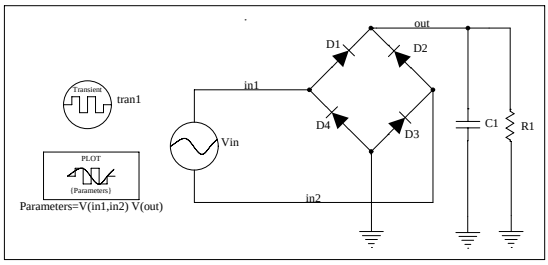


Figure 2-39 : Modélisation comportementale d'un pont de redressement double alternance.

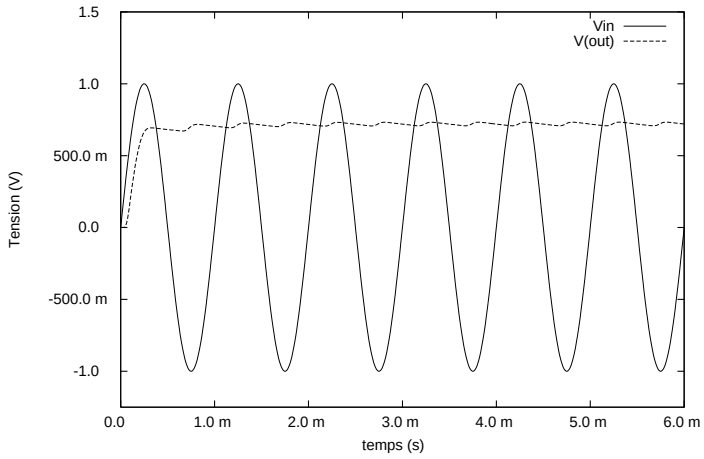


Figure 2-40 : Résultat de simulation d'un pont de redressement double alternance.

2.7.2.5 Transistor MOS correspondant au niveau 1 de modélisation SPICE

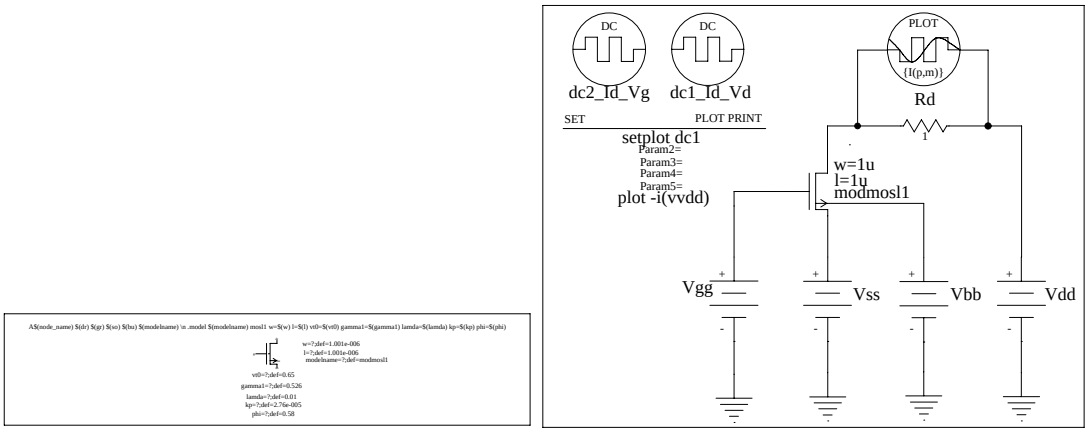
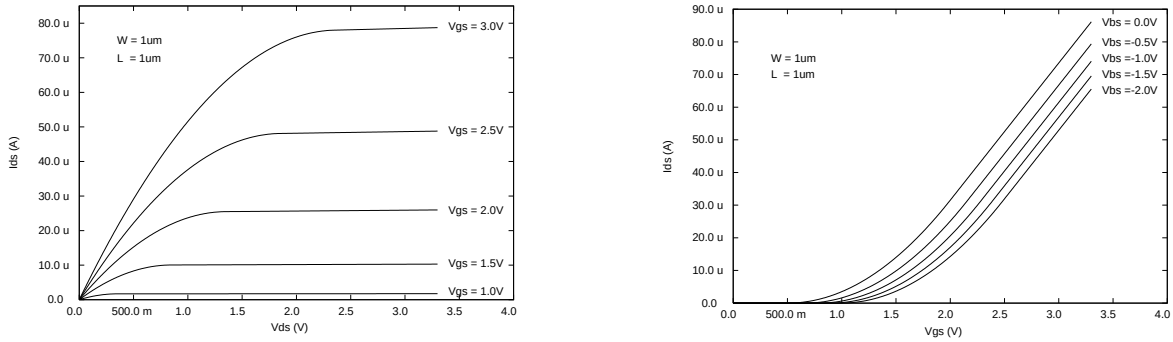


Figure 2-41 : Icône et schéma test d'un modèle VHDL-AMS de transistor MOS niveau 1.



a : Caractéristiques Ids = F(Vds) b : Caractéristique Ids = F(Vgs)

Figure 2-42 : Résultat de simulation de caractéristiques électriques de transistor MOS.

2.7.2.6 Interrupteur (*switch model*)

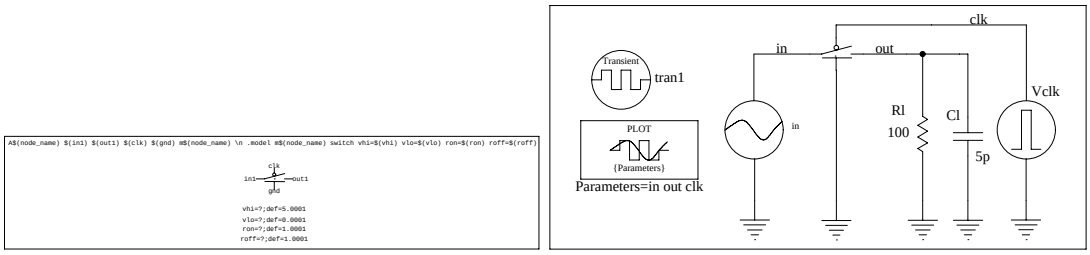


Figure 2-43 : Icône et schéma test d'un modèle VHDL-AMS d'interrupteur

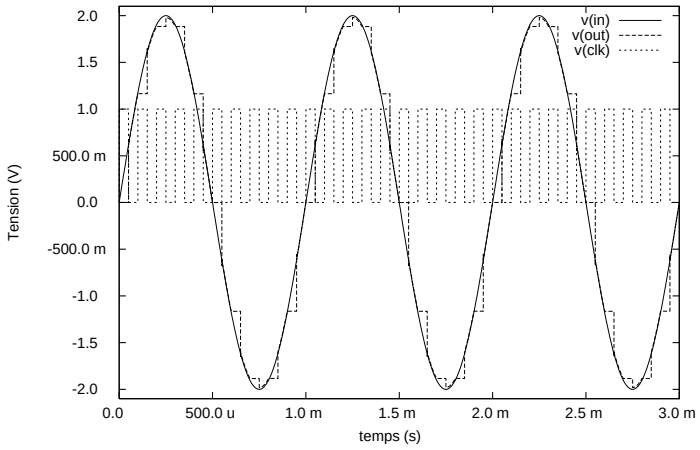


Figure 2-44 : Résultat de simulation d'un modèle d'interrupteur.

2.7.2.7 Convertisseur Flash

Convertisseur flash à 3 bits :

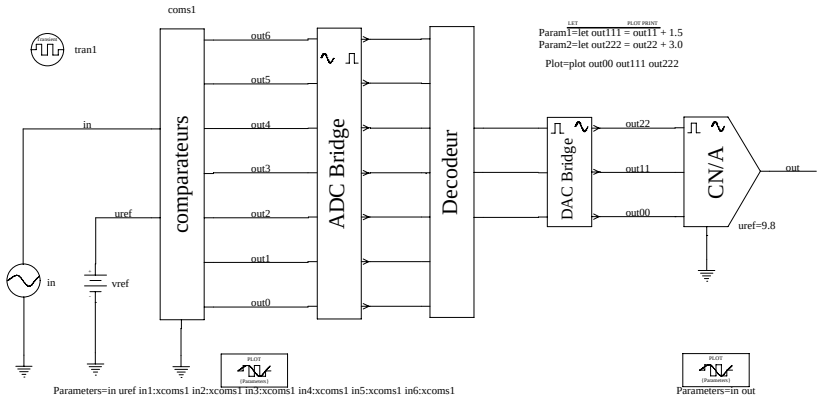


Figure 2-45 : Convertisseur A/N flash et N/A comportemental à base de VHDL-AMS et de XSPICE.

La tension à convertir est comparée à plusieurs seuils de référence ($(2^n - 1)$ comparateurs, avec n le nombre de bits à la sortie, nous avons donc 7 comparateurs). Le code obtenu est du type suivant (si nous comparons au niveau du $k^{ième}$ comparateur) :

A_{n-1}	A_{n-2}	…	A_k	A_{k-1}	…	A_1	A_0
0	0	0	0	1	1	1	1

Le décodeur réalise une logique combinatoire pour transformer ce mot en binaire (Figure 2-46).

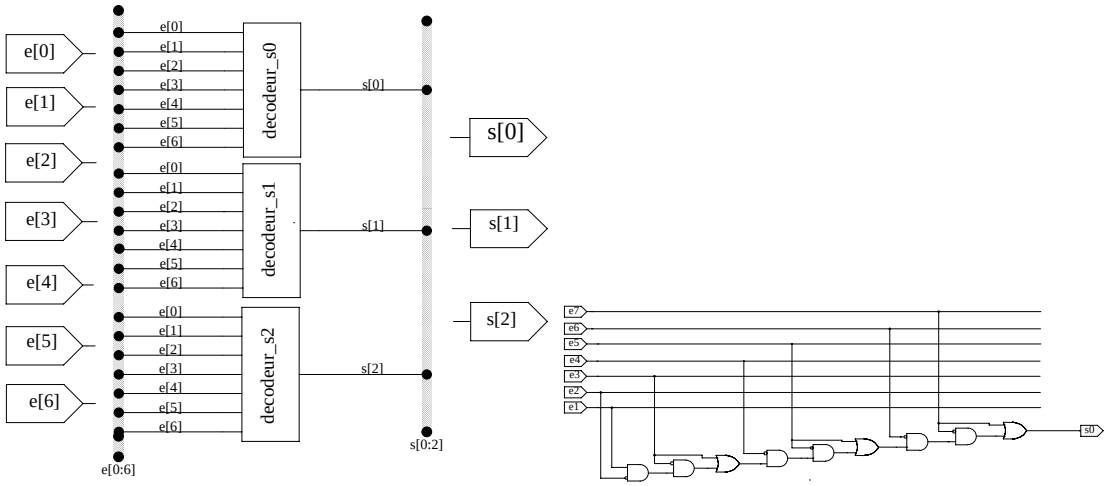


Figure 2-46 : Le décodeur à base de XSPICE.

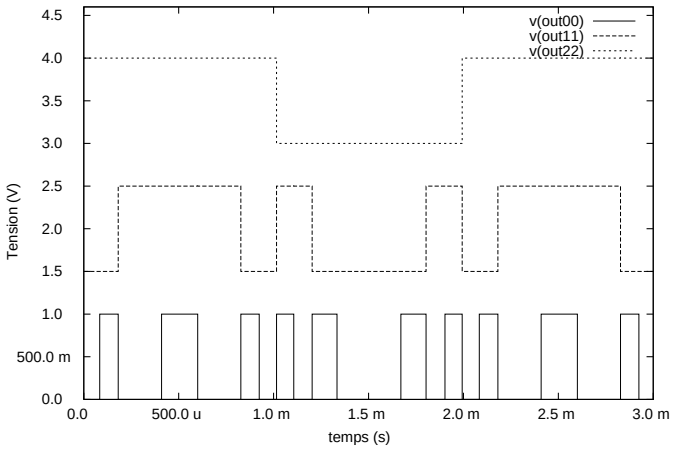


Figure 2-47 : Transformation de l'entrée qui est une sinusoïdal en mot binaire de 3 bits.

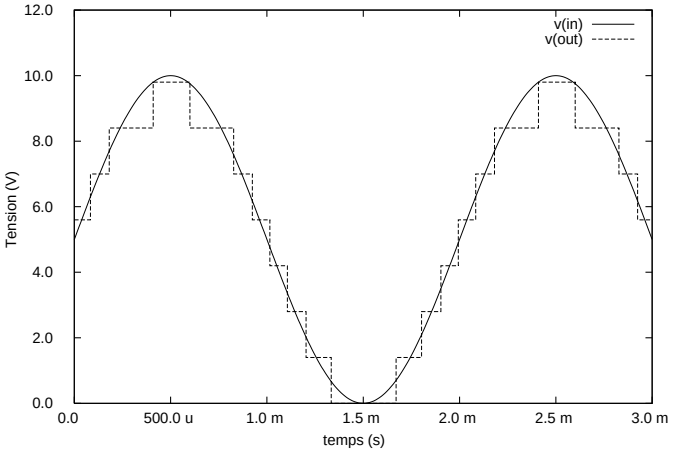


Figure 2-48 : L'entrée et la sortie après reconstitution du mot binaire par un CN/A.

2.7.3 Modélisation physique

2.7.3.1 Résistance

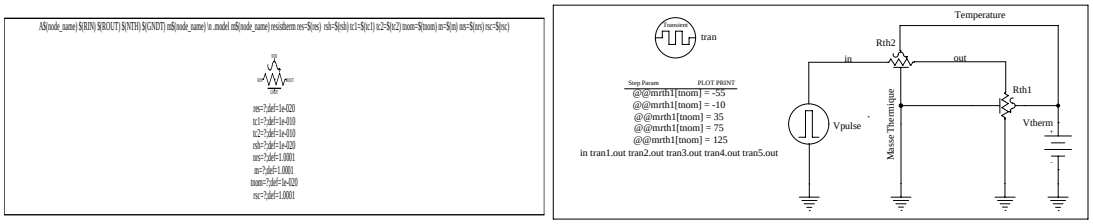


Figure 2-49 : (a) Icône d'un modèle de résistance physique

et (b) schéma test d'un diviseur de tension à base des résistances physique modélisé en VHDL-AMS.

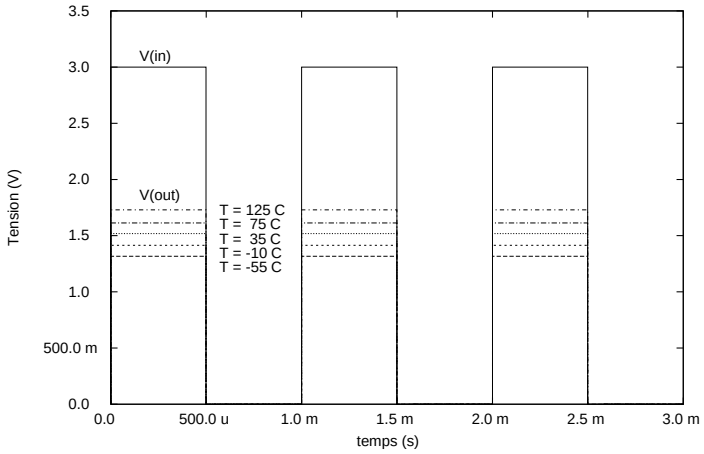


Figure 2-50: Résultat de simulation d'un diviseur de tension à base des résistances physique.

2.8 Conclusion

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

Grâce à VamSpiceDesigner, l'outil de modélisation et de simulation, nous avons élaboré et validé des modèles de différents types : fonctionnels, comportementaux & structurels, et physiques. Nous avons rangé tous ces modèles dans une bibliothèque multi-technologique pour la réutilisation.

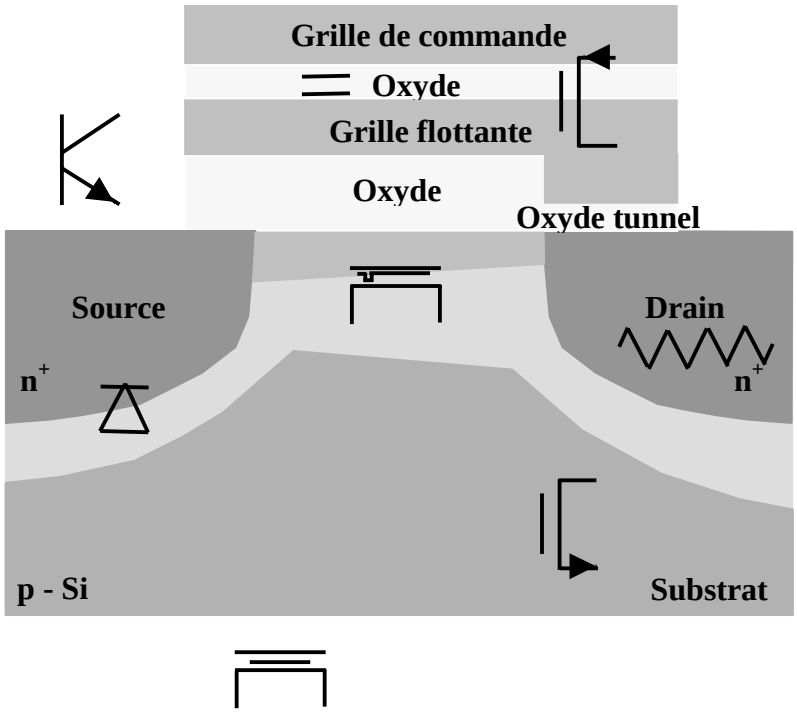
Avec cet outil, nous avons ainsi proposé une nouvelle méthodologie de conception multi-technologique (modélisation et simulation) basée sur une schématique hiérarchique.

Deuxième Partie

APPLICATIONS

3

MODELISATION-SIMULATION DE DISPOSITIFS MOS AVEC VHDL-AMS ET SPICE



Chapitre 3

3 MODELISATION – SIMULATION DE DISPOSITIFS MOS AVEC VHDL-AMS ET AVEC SPICE

3.1 Transistor MOS

3.1.1 Introduction

La plupart des modèles de dispositifs MOS présentent le modèle basé sur le schéma électrique équivalent suivant :

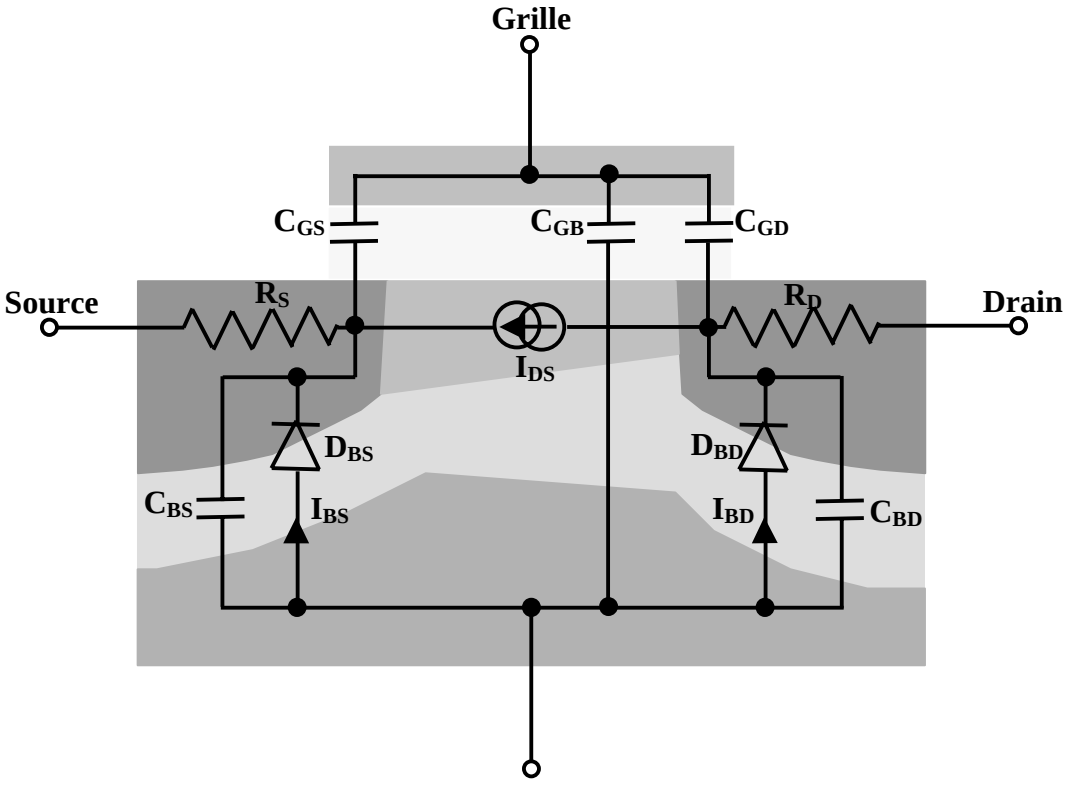


Figure 3-1 : Schéma électrique d'un transistor MOS.

Le schéma de la figure 3-1 est valable pour les dispositifs à canal N, pour le transistor MOS à canal P, on inverse la polarisation de tous les courants et de toutes les tensions.

Ce modèle comprend :

- Une source de courant statique drain-source I_{DS}, qui modélise le canal de conduction,
- 3 capacités relatives à la grille C_{GD}, C_{GS}, C_{GB}, qui modélisent les interactions entre charge de grille/charge de canal côté drain, charge de grille/charge de canal côté source et charge de grille/charge de substrat,
- 2 capacités relatives aux jonctions de substrat C_{BS} et C_{BD},
- 2 résistances d'accès côté source R_S et drain R_D,
- Pour les caractéristiques DC, 2 jonctions de substrat formant 2 diodes polarisées en inverse qui fournissent les courants I_{BS} et I_{BD} (certains modèles négligent ces éléments)

Dans tout ce qui suit les modèles sont basés sur ce schéma. Les paramètres en gras correspondent à des paramètres SPICE.

Le modèle SPICE du transistor MOS de niveau 1 a été proposé par C.T Sah. Les équations du modèle ont été ensuite modifiées pour être implantées dans le simulateur SPICE par H.Shichman et D.Hodges [SHI68][FOT97]. La caractéristique I_{DS} est modélisée dans les trois régions du domaine de fonctionnement du transistor :

La première région de faible inversion : $V_{GS} < V_{TH}$

On suppose qu'il n'existe pas de canal de conduction, d'où :

$$I_{DS} = 0 \tag{3.1}$$

Lorsque $V_{GS} > V_{TH}$, on distingue alors deux autres régions où l'expression de la caractéristique I_{DS} est une expression linéaire (quadratique) de V_{GS}; nous les appellerons donc régions linéaires et de saturation.

Pour la région linéaire ou quadratique : $V_{GS} \geq V_{TH}$ et $V_{DS} \leq V_{Dsat}$ avec $V_{Dsat} = V_{GS} - V_{TH}$

$$I_{DS} = \mathbf{KP}\left(\frac{W}{L - 2\,\mathbf{LD}}\right)\left(V_{GS} - V_{TH} - \frac{V_{DS}}{2}\right)V_{DS} \tag{3.2}$$

Pour la région de saturation : $V_{GS} \geq V_{TH}$ et $V_{DS} \geq V_{Dsat}$

$$I_{DS} = \frac{\mathbf{KP}}{2}\left(\frac{W}{L - 2\,\mathbf{LD}}\right)\left(V_{GS} - V_{TH}\right)^2 \tag{3.3}$$

Symbole du paramètre	Définition du paramètre	Unité
Paramètres de l'instance		
L	Longueur de canal	m
W	Largeur de canal	m
Paramètres du Processus de fabrication		
TOX	Epaisseur de l'oxyde sous la grille	m
LD	La réduction de longueur de canal par rapport à la valeur dessinée	m
NSUB	Dopage du substrat	AT/cm^3
Paramètres électriques		
UO	Mobilité des porteurs	$cm^2/(V.s)$
VTO	Tension de seuil à polarisation de substrat nulle	Volts
LAMBDA	Modulation de la longueur de canal	$Volts^{-1}$
KP	La transconductance	$A/Volts^2$
CGSO	Capacité grille source à polarisation nulle	F/m
CGDO	Capacité grille drain à polarisation nulle	F/m
CGBO	Capacité grille substrat à polarisation nulle	F/m

Tableau 3-1 : Les paramètres MOS du SPICE niveau 1 [FOT97].

Le modèle présenté précédemment est le plus simple ; 7 paramètres électriques suffisent pour caractériser son comportement électrique (Tableau 3-1). Mais la précision obtenue avec ce modèle est limitée à cause de nombreuses approximations faites. De plus les paramètres qui dépendent plus particulièrement du processus de fabrication sont peu nombreux. Aussi afin d'obtenir de meilleures caractéristiques I_{DS} en utilisant un nombre plus important de paramètres technologiques, il a été nécessaire de développer un modèle plus complet.

Le modèle de niveau 2 (LEVEL 2) se distingue du précédent par des expressions différentes du courant I_{DS} et des capacités C_{GS}, C_{GD} et C_{GB} qui prennent en compte des phénomènes plus fins (effet canal étroit, canal court, limitation de vitesse des porteurs…).

Le modèle SPICE niveau 2 [VLA80][MEY71] permettent la modélisation de la caractéristique I_{DS} dans trois zones de fonctionnement (voir tableau 3-2 pour les paramètres en gras):

Pour la région linéaire :

$$I_{DS} = \mu_s C_{OX} \frac{W}{L_{eff}} \left\{ \left(V_{GS} - V_{TH} - \frac{V_{DS}}{2} \right) - \frac{2}{3} f_s \, \mathbf{GAMMA} \left[\left(V_{DS} - V_{BS} + \mathbf{PHI} \right)^{\frac{3}{2}} - \left(\mathbf{PHI} - V_{BS} \right)^{\frac{3}{2}} \right] - f_n \left[\frac{V_{DS}^{\,2}}{2} + \left(\mathbf{PHI} - V_{BS} \right) V_{DS} \right] \right\} \quad (3.4)$$

Pour le régime de saturation :

$$I_{DS} = I_{DS\,sat}\,\frac{1}{1 - \mathbf{LAMBDA}\,V_{DS}} \tag{3.5}$$

Où $I_{DS\,sat}$ est la valeur du courant calculée pour $V_{DS} = V_{D\,sat}$.

En ce qui concerne la région de faible inversion :

En réalité une concentration d'électrons proche de la surface existe lorsque $V_{GS} < V_{TH}$, et par conséquent il existe un courant. Celui-ci est dû principalement aux phénomènes de diffusion entre la source et le canal.

Si la valeur du paramètre **NFS** (concentration des états de surface rapide) n'est pas renseignée, les phénomènes ne sont pas pris en compte :

$$I_{DS} = 0.0 \tag{3.6}$$

Sinon le modèle implanté dans SPICE introduit une dépendance exponentielle du courant I_{DS} par rapport à VGS pour la faible inversion. On définit la limite entre les régimes de faible et forte inversion par Von.

$$V_{on} = V_{TH} + n\frac{kT}{q} \tag{3.7}$$

$$\text{avec} \quad n = 1 + \frac{q\,\mathbf{NFS}}{C_{ox}} + \frac{1}{C_{ox}}\left[\frac{f_s\,\mathbf{GAMMA}\left(\mathbf{PHI} - V_{BS}\right)^{\frac{1}{2}} - f_n\left(\mathbf{PHI} - V_{BS}\right)}{2\left(\mathbf{PHI} - V_{BS}\right)}\right] \tag{3.8}$$

$$I_{DS} = I_{on}\,\exp\left(\frac{q\left(V_{GS} - V_{on}\right)}{nkT}\right) \tag{3.9}$$

Où Ion est la valeur du courant pour $V_{GS} = V_{on}$ dans la région de forte inversion et ainsi la continuité de la fonction I_{DS} est assurée. Cependant il existe une discontinuité des dérivées pour $V_{GS} = V_{on}$ ce qui pénalise la convergence de la simulation dans la région de transition.
Les expressions de f_s (les effets de canal court : *short factor*) et f_n (les effets de canal étroit : *narrow factor*):

$$f_S = 1 - \frac{\mathbf{XJ}}{2\,L_{eff}}\left\{\left[\left(1 + \frac{2W_{ds}}{\mathbf{XJ}}\right)^{\frac{1}{2}} - 1\right] + \left[\left(1 + \frac{2W_{dd}}{\mathbf{XJ}}\right)^{\frac{1}{2}} - 1\right]\right\} \tag{3.10}$$

$$W_{ds} = \left(\frac{2\varepsilon_{Si}\left(\mathbf{PHI} - V_{BS}\right)}{q\,\mathbf{NSUB}}\right)^{\frac{1}{2}} \qquad \text{et} \qquad W_{dd} = \left(\frac{2\varepsilon_{Si}\left(\mathbf{PHI} - V_{BS} + V_{DS}\right)}{q\,\mathbf{NSUB}}\right)^{\frac{1}{2}} \tag{3.11}$$

$$f_n = \mathbf{DELTA}\,\frac{\pi\,\varepsilon_{Si}}{4C_{ox}\,W} \tag{3.12}$$

L'expression de V_{TH} qui tient compte des effets dus à la géométrie du dispositif [FOT97] qui est négligé dans le modèle de niveau 1 :

$$V_{TH} = V_{fb} + \mathbf{PHI} + f_s\,\mathbf{GAMMA}\left(\mathbf{PHI} - V_{BS}\right)^{\frac{1}{2}} + f_n\left(\mathbf{PHI} - V_{BS}\right) \tag{3.13}$$

Dans l'expression de la caractéristique I_{DS} en régime linéaire intervient le paramètre L_{eff} qui correspond à la longueur effective du canal de conduction $L_{eff} = L - 2\,\mathbf{LD}$. En régime de saturation, il faut tenir compte de la longueur du canal et de la limitation de la vitesse **VMAX** des porteurs pour des champs électriques importants [FOT97].

Le modèle de niveau 1 considérait la mobilité comme constante. Cette approximation est en désaccord avec les résultats expérimentaux ; on observe une réduction de la mobilité lorsque la tension de grille augment. Afin de modéliser cet effet, on calcule μ_S par la relation suivante :

$$\mu_S = \mathbf{UO} \qquad E_S \le \mathbf{UCRIT} \tag{3.14}$$

$$\mu_S = \mathbf{UO}\left(\frac{\varepsilon_{Si}}{C_{ox}\left(V_{GS} - V_{TH} - \mathbf{UTRA}\,V_{DS}\right)}\right)^{\mathbf{UEXP}} \qquad E_S > \mathbf{UCRIT} \tag{3.15}$$

On voit que l'expression de μ_S, nouvelle mobilité, tient compte des effets de dégradation dus à l'accroissement de l'intensité des champs électriques dans la région proche de l'interface semi-conducteur/oxyde. Elle dépend des paramètres **UCRIT**, **UTRA** et **UEXP** ; **UCRIT** caractéristique du champ électrique transversal critique, **UTRA** est un coefficient numérique, **UEXP** un exposant empirique. Cette relation permet d'obtenir une bonne correspondance entre les simulations SPICE et les résultats expérimentaux dans la région de forte inversion, sauf pour des transistors à canal court et étroit.

Symbole du paramètre	Définition du paramètre	Unité
Paramètres de l'instance		
L	Longueur de canal	M
W	Largeur de canal	m
Paramètres du Processus de fabrication		
TOX	Epaisseur de l'oxyde sous la grille	m
LD	La réduction de longueur de canal par rapport à la valeur dessinée	m
WD	La réduction de largeur de canal par rapport à la valeur dessinée	m
Paramètres électriques		
VTO	Tension de seuil à polarisation de substrat nulle	Volts
UO	Mobilité des porteurs	$cm^2/(V.s)$
UCRIT	Coeff pour la mobilité pour un champ critique	V/cm
UEXP	Coeff pour la mobilité pour un champ critique (exposant)	-
DELTA	Effet de largeur de canal sur la tension de seuil	-

NSUB	Dopage du substrat	AT/cm^3
XJ	Profondeur de la diffusion	m
VMAX	la limitation de la vitesse des porteurs pour des champs électriques importants	m/s
NEFF	Coeff pour la charge totale (mobile et fixe) dans le canal	-
NFS	Densité d'état de surface rapide	AT/cm^2
ETA	Effet statique	-
GAMMA	Effet de la polarisation de substrat sur la tension de seuil	Volts$^{1/2}$
PHI	Potentiel de la surface dans la région de forte inversion	Volts
LAMBDA	Modulation de la longueur de canal	Volts^{-1}
KP	La transconductance	A/Volts^2
UTRA	Coeff pour la mobilité pour un champ critique (champ transverse)(HSPICE)	-

Tableau 3-2 : Les paramètres SPICE du transistor MOS LEVEL 2.

3.1.2 Modèle SPICE niveau 3 du transistor MOS

Le modèle niveau 3 (LEVEL 3) a été développé pour simuler les transistors MOSFET à canal court ; les caractéristiques obtenues sont assez précises lorsque le dispositif a une longueur ou une largeur de canal inférieure à 2µm.

Les expressions ont été proposées par Dang [FOT97][DAN79] et adaptées au simulateur SPICE par Vladimirescu [VLA80], celles pour la faible inversion sont empruntées au modèle LEVEL 2.

Cependant, les différences fondamentales résident dans l'expression de la caractéristique I_{DS} pour la région linéaire. Pour conserver toute sa précision, il introduit dans l'équation des expressions additionnelles empiriques qui permettent de prendre en compte les effets de canal étroit. Le modèle est ainsi composé d'un ensemble de relations de base simplifiées par l'utilisation de paramètres empiriques; elles permettent d'améliorer la précision du modèle.

Comme le modèle de niveau 2, le modèle de niveau 3 permet la modélisation de la caractéristique I_{DS} dans trois zones de fonctionnement [FOT97][ANT88]: (Pour les définitions des paramètres, voir Annexe 1)

Régime linéaire : $0 < V_{DS} < V_{DS\,sat}$ et $V_{GS} > V_{TH}$

$$I_{DS} = \mu_S C_{ox} \frac{W}{L_{eff}} \left(V_{GS} - V_{TH} - \frac{(1+f_B)V_{DS}}{2} \right) V_{DS} \qquad (3.16)$$

Régime de saturation : $V_{DS} > V_{DS\,sat}$ et $V_{GS} > V_{TH}$

$$I_{DS} = \mu_S C_{ox} \frac{W}{L_{eff}} \left(V_{GS} - V_{TH} - \frac{(1+f_B)V_{Dsat}}{2} \right) V_{Dsat} \qquad (3.17)$$

$$\text{avec } f_B = \frac{\mathbf{GAMMA}\, f_S}{4\, f(V_{BS})} + f_n \tag{3.18}$$

Où les paramètres empiriques f_s et f_n ont été introduits pour modéliser :

- f_s les effets de canal court (*short factor*)
- f_n les effets de canal étroit (*narrow factor*)

La fonction $f(V_{BS})$ dépend du potentiel de surface où VBS est de la tension substrat-source. En fonctionnement normal, la jonction substrat source est polarisée en inverse ($V_{BS} \leq 0$) et ($V_{BS} > 0$) dans le cas où cette jonction est conductrice.

Régime de faible inversion : $V_{GS} < V_{TH}$

Comme pour le niveau 2, ces phénomènes ne sont pris en compte que lorsque **NFS** n'est pas nul.
Si **NFS** = 0,

$$I_{DS} = 0.0 \tag{3.19}$$

Sinon on définit la limite entre les régimes de faible et de forte inversion par V_{on}.

$$V_{on} = V_{TH} + n\frac{kT}{q} \tag{3.20}$$

$$\text{avec : } n = 1 + q\frac{\mathbf{NFS}}{C_{ox}} + \frac{\mathbf{GAMMA}\, f_S\, f(V_{BS}) + f_n\,(\mathbf{PHI} - V_{BS})}{2\,(\mathbf{PHI} - V_{BS})} \tag{3.21}$$

$$I_{DS} = I_{on}\, \exp\left(\frac{q\,(V_{GS} - V_{on})}{nkT}\right) \tag{3.22}$$

Où Ion est la valeur du courant pour $V_{GS} = V_{on}$ dans la région de forte inversion et ainsi la continuité de la fonction I_{DS} est assurée. Cependant il existe une discontinuité des dérivées pour $V_{GS} = V_{on}$ ce qui pénalise la convergence de la simulation dans la région de transition.

Les expressions des capacités relatives à la grille ont été proposées par Meyer [MEY71]. Dans ce modèle simplifié, les effets dus à l'accumulation de charge dans la partie intrinsèque du transistor sont modélisés par 3 capacités non-linéaires : C_{GB}, C_{GS} et C_{GD}. Cependant les expressions de Meyer n'étaient pas continues au passage des différentes régions de fonctionnement ; dans le modèle SPICE LEVEL 1, celles-ci ont été adaptées par Shichann et Hodges en les modifiant par des facteurs multiplicatifs dépendant de la polarisation qui assurent la continuité des expressions.

La valeur des capacités associées à la région désertée entre le drain et le substrat C_{BD}, et entre la source et le substrat C_{BS} dépend respectivement de la tension aux bornes des jonctions pn drain-substrat et source-substrat polarisées en inverse. En examinant la géométrie de la région désertée, la capacité de déplétion est la combinaison d'une capacité plate C_j (Figure 3-2) et d'une capacité qui fait intervenir les cotés des différentes diffusions C_{jsw}.

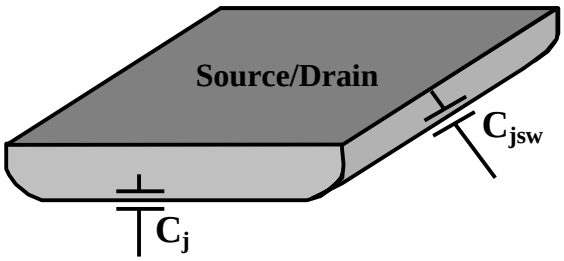
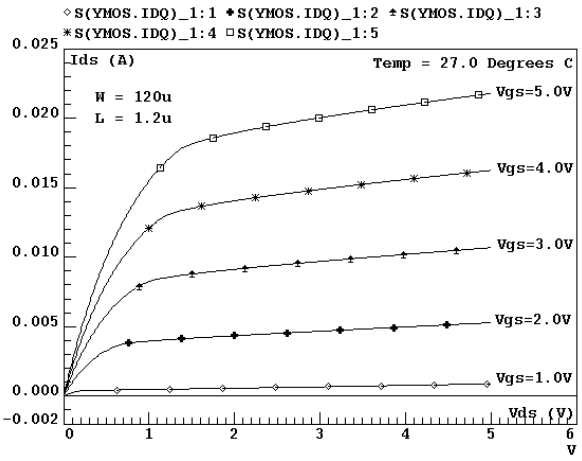


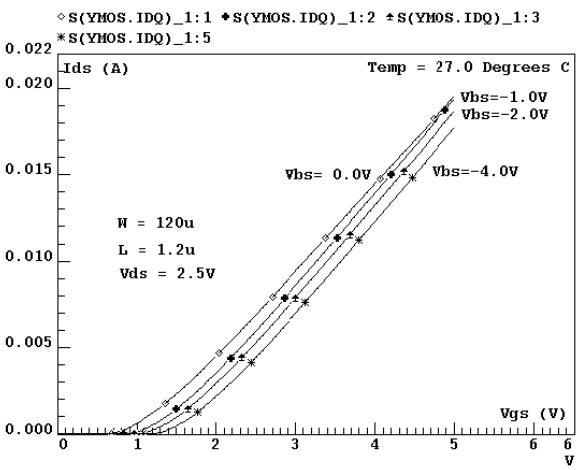
Figure 3-2 : Décomposition de Source/Drain en deux capacités de jonction : une capacité « plate » C_j et une capacité faisant intervenir les cotés C_{jsw}

3.1.3 Implantation VHDL-AMS du modèle SPICE niveau 3 (voir annexe 1)

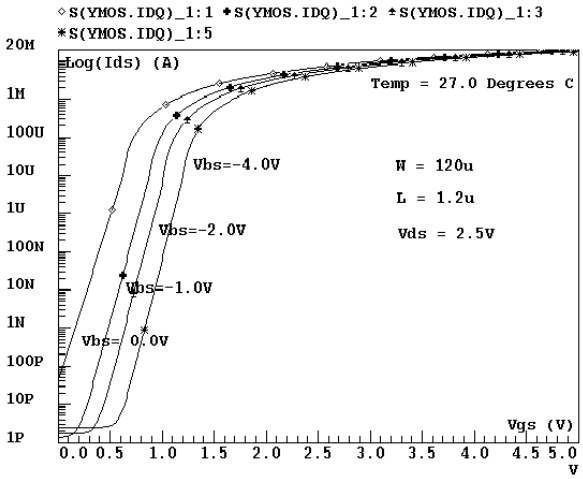
La figure 3-3-a montre les caractéristiques de I_{DS} en fonction de V_{DS} en faisant varier V_{GS} de 1 à 5V. Les figures 3-3-b et 3-3-c sont les caractéristiques de I_{DS} et son logarithme en fonction de V_{GS} en faisant varier V_{BS} de –4 à 0V. On constate sur la figure 3-3-c que le courant sous-seuil de MOS3 est raccorder entre le faible et la forte inversion. La figure 3-3-d montre la transconductance g_m en fonction de V_{gs} en variant V_{BS}. La transconductance est définie par : $g_m = \left.\frac{\partial I_{DS}}{\partial V_{GS}}\right)_{V_{DS}=C^{te}}$. Les figure 3-3-e et 3-3-f sont les caractéristiques de g_{ds} et son logarithme en fonction V_{DS} (1 à 5V). La deuxième courbe montre mieux la discontinuité de g_{ds}. la conductance de sortie g_{ds} est définie par : $g_{ds} = \left.\frac{\partial I_{DS}}{\partial V_{DS}}\right)_{V_{GS}=C^{te}}$.



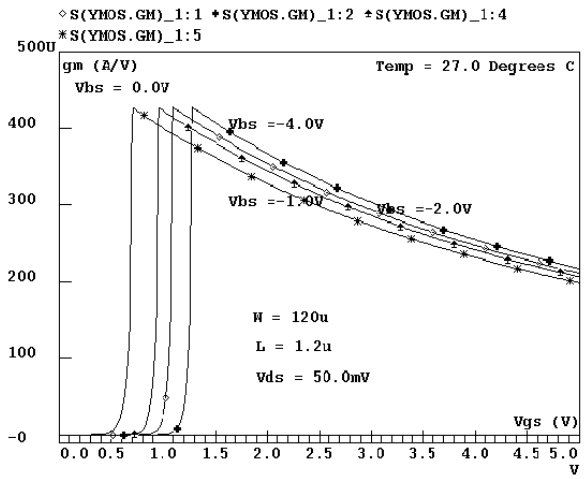
a : Caractéristiques $I_{DS} = F(V_{DS})$.



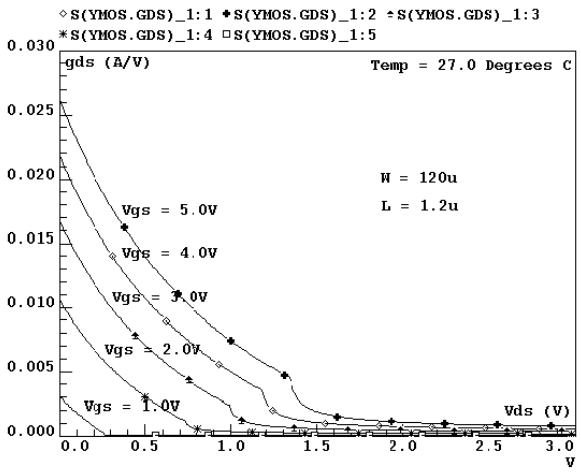
b : Caractéristiques $I_{DS} = F(V_{GS})$



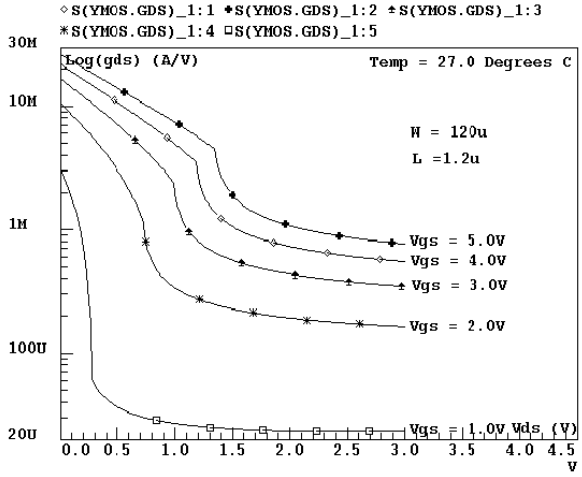
c : Caractéristiques
$Log(I_{DS}) = F(V_{GS})$.



d : Caractéristiques
$g_m = F(V_{GS})$.



e : Caractéristiques
$g_{ds} = F(V_{DS})$.



f : Caractéristiques
$Log(g_{ds}) = F(V_{DS})$.

Figure 3-3 : Caractéristiques statiques d'un transistor MOS niveau 3 modélisé en VHDL-AMS.

3.2 Cellule mémoire de type EEPROM

3.2.1 Introduction

Les mémoires à semi-conducteur peuvent être classées (Figure 3-4) en deux grandes familles; les mémoires volatiles et les mémoires non-volatiles. Ce découpage est basé sur leur capacité de rétention de l'information après interruption de toute alimentation.

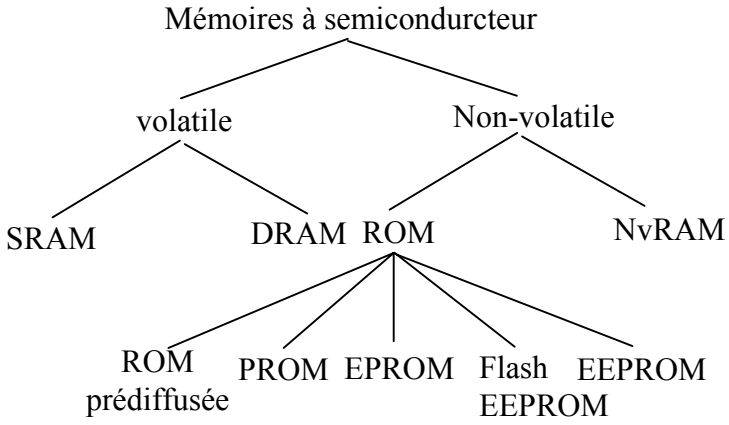


Figure 3-4 : Les grandes familles de mémoires.

Tandis que la mémoire volatile perd son contenu hors alimentation électrique, la mémoire non-volatile le conserve. Une des principales applications des mémoires non-volatile est la sauvegarde du programme de lancement d'ordinateur.

Les mémoires à semi-conducteur peuvent être fabriquées soit en technologie bipolaire, soit en technologie MOS. La technologie MOS offre l'avantage d'une forte densité d'intégration, et de meilleurs comportements électriques, les mémoires bipolaires sont cependant plus rapides. Dans cette partie, on va étudier que les mémoires en technologies MOS [BEN99].

3.2.2 Modélisation de la cellule EEPROM

Le point mémoire est constitué de deux transistors (Figure 3-5) d'état et de sélection. Le premier (T1) est un transistor MOS de haute tension. Il permet, dans une matrice mémoire (Figures 3-5 et 3-6), la sélection du second (T2). Celui-ci, disposant d'une grille flottante, représente le transistor d'état.

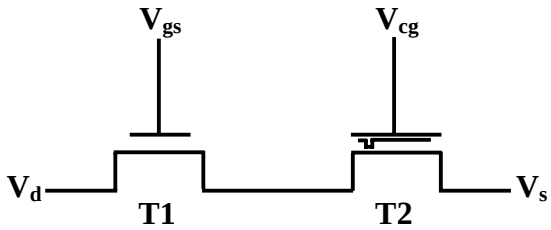


Figure 3-5 : Structure électrique de la cellule EEROM.

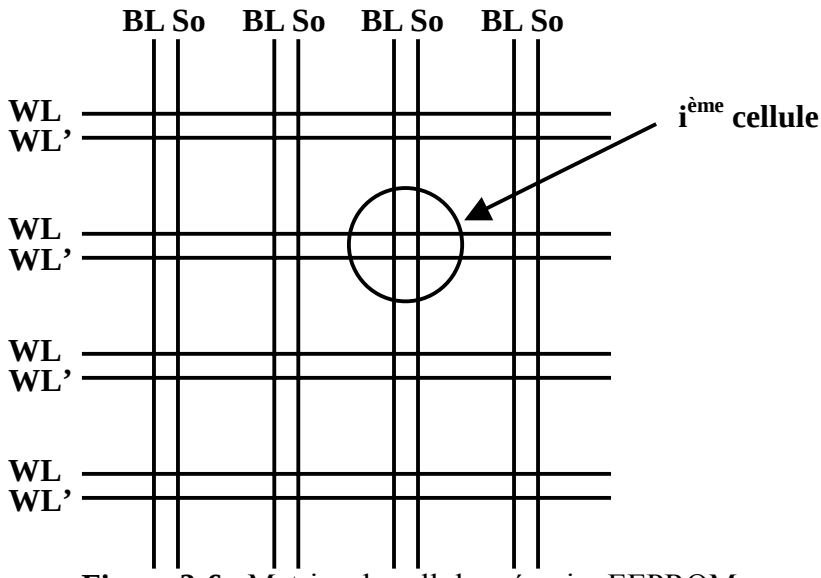


Figure 3-6 : Matrice de cellule mémoire EEPROM.

Où WL est la *word line* (grille de sélection), WL' est la grille de contrôle (V_{gc}) alors que BL est la *bit line* (drain de la cellule) (V_d) et So la source de cellule (V_s).

3.2.2.1 Transistor de sélection

Nous appliquons une haute tension ($V_{gs} \approx 16V$) sur la grille pour sélectionner ce transistor. Ce dernier fonctionne, donc, comme un interrupteur à deux états : état haut et état bas (Figure 3-21).

Nous utilisons le modèle de transistor MOS de SPICE LEVEL 3 décrit dans la section précédente. En effet, ce modèle est assez complet et précis pour étudier l'effet de l'auto échauffement par exemple.

3.2.2.2 Transistor d'état

Il y a une large variété de mémoires non-volatiles. Chaque type de mémoire a ces caractéristiques liées à la structure de leur cellule de base.

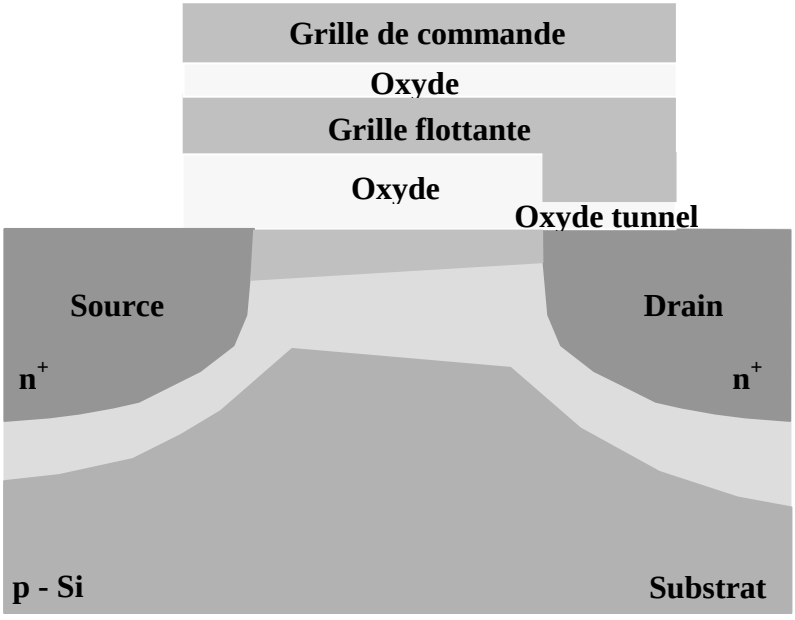


Figure 3-7 : Structure d'un MOSFET à grille flottante.

Dans notre cas, nous étudions le transistor MOS à une seule grille flottante (FG). Celle-ci est réalisée en polysilicium (Figure 3-7). Pour améliorer les opérations de programmation (écriture) et d'effacement, une grille additionnelle est placée au-dessus de la FG qui est appelée grille de commande (CG). La FG agit comme un puits de potentiel. Si une charge y pénètre, elle ne peut en sortir que par l'application d'une force extérieure. La FG stocke les charges.

La modélisation du transistor d'état ou encore transistor MOS à grille flottante peut utiliser la même démarche que celle suivie lors de la modélisation d'un transistor MOS. En effet, nous prenons la grille conventionnelle comme une grille flottante et celle-ci est séparée du drain par une couche fine d'oxyde (Figure 3-8).

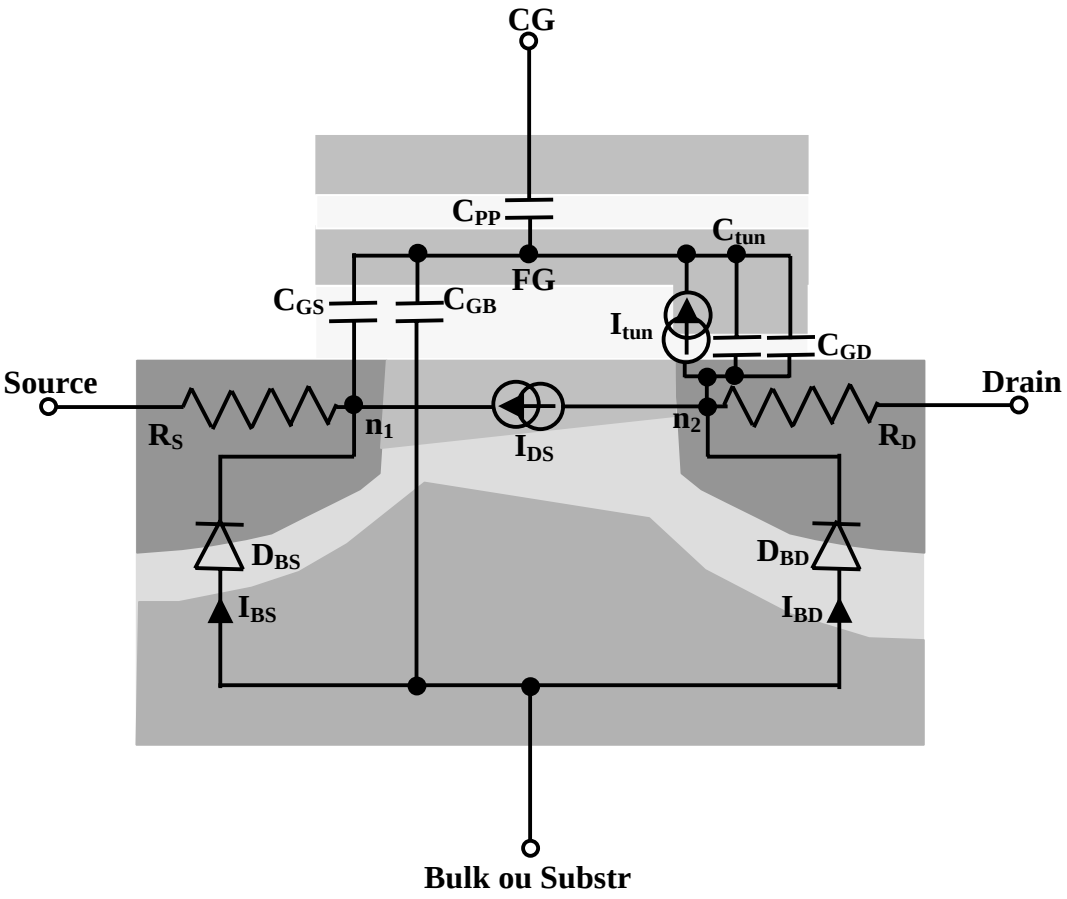


Figure 3-8 : Schéma électrique d'un transistor EEPROM.

Les diffusions Source et Drain

Les diffusions source et drain sont représentées par deux résistances, respectivement R_s et R_d. On peut utiliser les mêmes résistances décrites dans le modèle SPICE LEVEL 3.

Jonctions PN

Le transistor MOS présente deux jonctions PN reliant le drain et la source au substrat (Figure 3-8) ces deux jonctions forment deux diodes.

$$I_{BS} = \mathbf{IS}\left[\exp\left(-\frac{\left(V_{n1} - V_b\right)}{U_t}\right) - 1\right] \qquad (3.23)$$

I_{BS} (I_{BD}) est le courant traversant la diode formée par les jonctions source-substrat (drain-substrat). IS est le courant de saturation de la diode alors que V_{n1} (V_{n2}) est le potentiel interne côté source (côté drain).

Modèle capacitif

Le modèle capacitif simplifié du transistor (Figure 3-8) est un peu modifié puisque la capacité représentant la zone d'injection se trouve de plus par rapport au transistor de sélection qui est en parallèle avec la capacité de Meyer C_{gd} [MEY71][ANT88].

Les capacités C_{gs}, C_{gd} et C_{gb} sont évaluées en fonction du régime de fonctionnement. Nous utilisons les capacités qui sont décrites dans le LEVEL 3 mais nous devons changer leurs références c'est-à-dire qu'au lieu d'être référencées par rapport à la grille de contrôle, elles seront référencées par rapport à la grille flottante ($V_{gs} - V_{th}$) = ($V_{fgs} - V_{TH}$) côté source ; il en sera de même côté drain.

La capacité tunnel C_{tun} de type nMOS est non linéaire ; elle peut être modélisée suivant deux régions :

Régime d'accumulation :

$$C_{tun} = C_{tun\ max} \tag{3.24}$$

Régime de désertion :

$$C_{tun} = \frac{C_{tun\ max}}{\left(1 + \frac{V_a}{\mathbf{VJ}}\right)^{\mathbf{MJ}}} \tag{3.25}$$

Avec :

$$C_{tun\ max} = \frac{\varepsilon_0 \varepsilon_{ox} S_{tun}}{T_{tun}} \tag{3.26}$$

S_{tun} est la surface de la couche tunnel et T_{tun} est son épaisseur. ε_0 est la permittivité du vide et ε_{ox} est celle de l'oxyde. $C_{tun\ max}$ est la valeur maximale de la capacité C_{tun}. VJ et MJ sont respectivement la barrière de potentiel et le facteur de gradualité. Va est la tension appliquée aux bornes de la capacité.

La présence de la grille de contrôle dans cette structure a une influence importante. Celle-ci est séparée de la grille flottante par une épaisse couche d'oxyde qui est modélisée par la capacité C_{pp}. La grille de contrôle, dans la technologie simple polysilicium, est dopée N. Ceci permet de considérer que la capacité C_{pp} est de type nMOS ayant la même caractéristique que la capacité C_{tun}. De même S_{PP} (S_{tun}) est la surface de la couche d'oxyde entre la grille flottante et la grille de contrôle. Le paramètre T_{PP} (T_{tun}) est son épaisseur.

Cette nouvelle capacité est à l'origine de la variation de l'équation régissant la conservation de charge dans le modèle capacitif de la structure du transistor MOS à grille flottante (Figure 3-9). L'équation 3.28 permet de déduire l'expression de la charge de la grille flottante qui peut être positive ou négative suivant le signe de la différence de potentiel ($V_{ds} - V_{fgs}$).

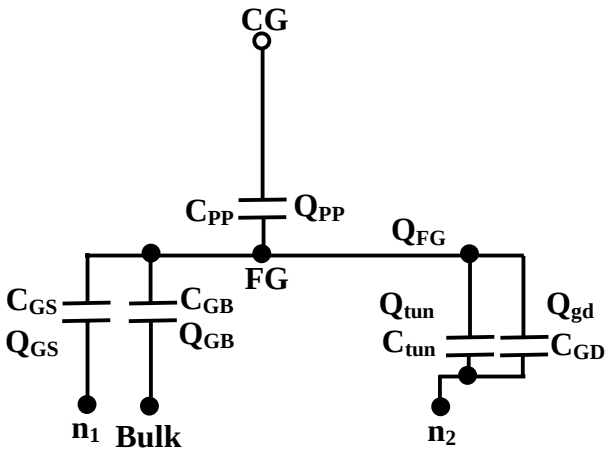


Figure 3-9 : Modèle capacitif du transistor du transistor MOS à grille flottante.

$$\begin{aligned} Q_{PP} &= C_{PP}\left(V_{fg} - V_{cg}\right) \\ Q_{tun} &= C_{tun}\left(V_{fg} - V_{n2}\right) \\ Q_{gd} &= C_{gd}\left(V_{fg} - V_{n2}\right) \\ Q_{gs} &= C_{gs}\left(V_{fg} - V_{n1}\right) \\ Q_{gb} &= C_{gb}\left(V_{fg} - V_{b}\right) \end{aligned} \tag{3.27}$$

$$Q_{fg} = \Sigma Q_i \Rightarrow Q_{fg} = Q_{PP} + Q_{tun} + Q_{gs} + Q_{gd} + Q_{gb} + Q_{fg0} \tag{3.28}$$

La charge Q_{fg0} est la charge initiale existante dans la grille flottante.

La tension V_{fg} obtenue au niveau de la grille flottante (Equation 3.29) est déduite de l'expression de la neutralité électrique (Equation 3.28).

$$V_{fg} = \frac{Q_{fg}}{C_{tot}} + \frac{C_{PP}}{C_{tot}} V_{cg} + \frac{C_{tun}}{C_{tot}} V_{n2} + \frac{C_{gs}}{C_{tot}} V_{n1} + \frac{C_{gd}}{C_{tot}} V_{n2} + \frac{C_{gb}}{C_{tot}} V_b \tag{3.29}$$

Avec :

$$C_{tot} = C_{PP} + C_{tun} + C_{gb} + C_{gs} + C_{gd} \tag{3.30}$$

Nous posons

$$K_{eff} = \frac{C_{PP}}{C_{tot}} \quad \text{et} \quad K_{ecr} = \frac{C_{tun}}{C_{tot}} \tag{3.31) et (3.32}$$

K_{eff} représente le coefficient de couplage entre la grille de contrôle V_{cg} et celle de la grille flottante V_{fg} et caractérise les performances de la cellule durant la phase d'effacement. Quant à K_{ecr}, il représente le coefficient de couplage entre la tension de drain Vd et la tension de la grille flottante

V_{fg}. Ce coefficient est prépondérant durant la phase d 'écriture. L'influence de ces deux coefficients montre l'importance des deux capacités C_{PP} et C_{tun} durant les opérations d'écriture et d'effacement.

L'expression obtenue de la grille flottante permet d'extraire celle de la tension de seuil du transistor. Il est tout de même à signaler qu'il faut faire la différence entre la tension de seuil V_{tgc} du transistor vue de la grille de contrôle et la tension de seuil V_t vue de la grille flottante. Ainsi, nous pouvons considérer que $V_{tgc} = V_{cg}$ lorsque $V_{fg} = V_t$. L'expression de la tension de seuil au niveau de la grille flottante V_t et celle du transistor d'état au niveau de la grille de contrôle V_{tgc} sont, donc, déduites à partir des équations 3.33 et 3.34.

$$V_{TH} = \frac{Q_{fg}}{C_{tot}} + K_{eff}\, V_{tcg} + K_{ecr}\, V_{n2} + \frac{C_{gd}}{C_{tot}} V_{n2} + \frac{C_{gs}}{C_{tot}} V_{n1} + \frac{C_{gb}}{C_{tot}} V_b \tag{3.33}$$

$$V_{tcg} = \frac{V_{TH}}{K_{eff}} - \frac{Q_{fg}}{C_{PP}} - \frac{K_{ecr}}{K_{eff}} V_{n2} - \frac{C_{gd}}{C_{PP}} V_{n2} - \frac{C_{gs}}{C_{PP}} V_{n1} - \frac{C_{gb}}{C_{PP}} V_b \tag{3.34}$$

L'équation 3.34 représente la tension de seuil utilisée pendant la lecture de l'état de la cellule. En effet, la tension de seuil effective est évaluée dés que le canal de courant drain commence à se former ($V_d \approx V_s = 0$) et ne dépend, donc, que de la charge injectée dans la grille flottante et des capacités du transistor d'état. Ainsi, la différence de tension de seuil peut être exprimée par :

$$V_{tcg} = \frac{V_{TH}}{K_{eff}} - \frac{Q_{fg}}{C_{PP}} \tag{3.35}$$

Générateurs de courant IDS et Itun

Les équations proposées dans le modèle SPICE LEVEL 1 sont utilisées. Le paramètre ($V_{fg} - V_{TH}$) existant dans les expressions du courant drain en régime linéaire et en saturation (voir LEVEL 1) est donc remplacé par K_{eff} ($V_{cg} - V_{tcg}$) ou en tient compte du couplage de la grille de commande par rapport à la grille flottante. Les équations finales du courant drain dans un transistor MOS à grille flottante sont données par :

Régime linéaire : Si $V_{cgs} > V_{tcg}$ et $V_{ds} < K_{eff}(V_{cgs} - V_{tgc})$

$$I_{DS} = K_{eff}\, \mu_n\, C_{ox}' \frac{W}{L_{eff}} \left((V_{cgs} - V_{tcg}) V_{ds} - \frac{V_{ds}^2}{2\, K_{eff}} \right) \tag{3.36}$$

Régime saturé : Si $V_{cgs} > V_{tcg}$ et $V_{ds} > K_{eff}(V_{cgs} - V_{tgc})$

$$I_{DS} = K_{eff}^2\, \mu_n\, C_{ox}' \frac{W}{2\, L_{eff}} (V_{cgs} - V_{tcg})^2 \tag{3.37}$$

avec $L_{eff} = L - 2\, LD$

Nous remarquons que la même approche est adoptée pour recalculer, à partir des équations de C_{gs}, C_{gd} et C_{gb} les nouvelles expressions des capacités du transistor en régime d'accumulation, de désertion, linéaire et saturé. Pour les équations des capacités de Meyer, voir annexe 2.

L'expression du générateur de courant tunnel de type Fowler-Nordheim (FN), en négligeant l'effet de la température, est donnée par :

$$I_{tun} = \mathbf{Stun}\,\alpha\, E_{tun}^{\,2} \exp\left(-\frac{\beta}{E_{tun}}\right) \tag{3.38}$$

Le champ électrique E_{tun} aux bornes de la zone d'injection peut être exprimé par la différence de potentiel $V_{fgs} - V_{ds}$ diviséepar l'épaisseur de la couche d'oxyde Ttun.

$$E_{tun} = \frac{V_{fgs} - V_{ds}}{\mathbf{Ttun}} \tag{3.39}$$

Stun est la surface de la couche tunnel. α et β sont les paramètres Fowler-Nordheim :

$$\alpha = \frac{q^3\, m}{8\pi\, h\, q\, \Phi_{bo}\, m^*} \tag{3.40}$$

$$\beta = \frac{8\pi \sqrt{2\, m^*} \left(q\, \Phi_{bo}\right)^{3/2}}{3\, q\, h} \tag{3.41}$$

$q = 1.6\ 10^{-19}$ C : charge de l'électron,
$m = 9.1\ 10^{-31}$ Kg : masse de l'électron,
$h = 6.62\ 10^{-34}$ J.s : constante de Planck,
$\Phi_{bo} = 3.13$ eV : barrière de potentiel,
$m^*_e = 0.33$ m : masse effective de l'électron,
$m^*_t = 0.54$ m : masse effective du trou.

Leurs valeurs dans le cas de l'effacement et de l'écriture [BEN99]:

ALPHAEFF = $1.489\ 10^{-6}$ A V^{-2}
BETAEFF = $21.72\ 10^{9}$ V m^{-1}
ALPHAECR = $0.91\ 10^{-6}$ A V^{-2}
BETAECR = $27.78\ 10^{9}$ V m^{-1}

Regroupant toutes les structures utilisées dans cette première partie, le modèle complet du transistor considéré est donné par la figure 3-10 :

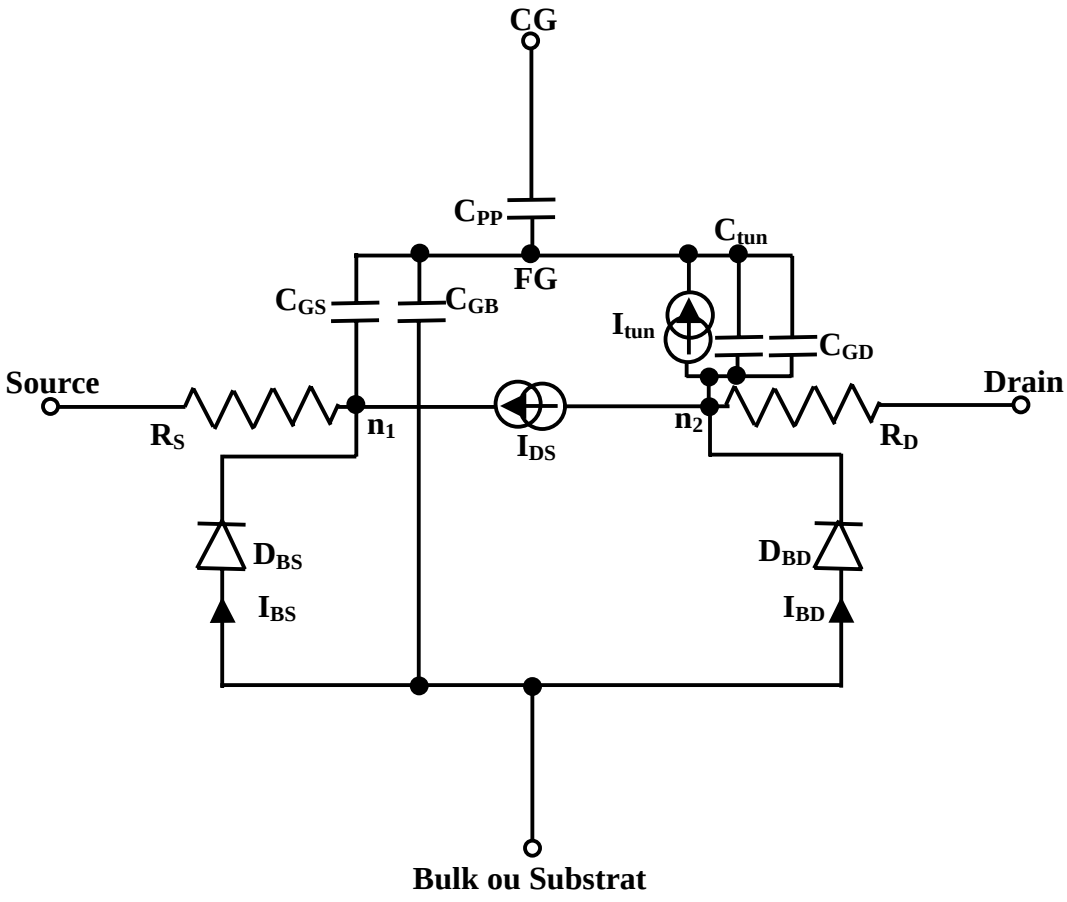


Figure 3-10 : Modèle complet du transistor MOS avec la zone tunnel.

Symbole du paramètre	Définition du paramètre	Unité	Valeur par défaut
Paramètres liés au dispositif			
L	Longueur de canal	m	1.5u
W	Largeur de canal	m	2u
Paramètres du modèle			
GAMMA	Effet de la polarisation de substrat sur la tension de seuil	$V^{1/2}$	0.0
PHI	Potentiel de la surface dans la région de forte inversion	Volt	0.6
TOX	Epaisseur de la couche d'oxyde	m	30.0n
TPP	Epaisseur de la couche d'oxyde entre grille de contrôle et grille flottante	m	26.5n
TTUN	Epaisseur de la couche d'oxyde tunnel	m	8.8n
SPP	Surface de la zone formée par la grille flottante et la grille de contrôle	m^2	13.3u
STUN	Surface de la zone tunnel	m^2	0.64u
QFGO	Charge initiale de la grille flottante	C	0.0
NSUB	Dopage du substrat	AT/cm^3	0.0
MUO	Mobilité des porteurs	$cm^2/(V.s)$	260
LD	Diffusion latérale	m	500n
RS	Résistance de la source	Ohm	50.0

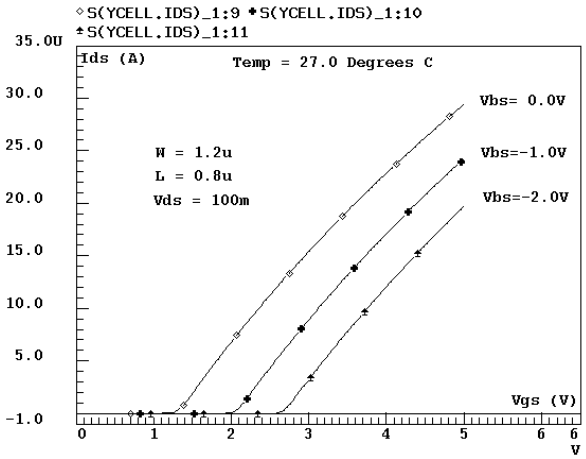
RD	Résistance du drain	Ohm	40.0
IS	Courant de saturation des diodes	A	26.0f
CGBO	Capacité de débordement G-B	fF/µm	$6.3\ 10^{-3}$
CGDO	Capacité de débordement G-D	fF/µm	$575.2\ 10^{-3}$
CGSO	Capacité de débordement G-S	fF/µm	$575.2\ 10^{-3}$
ALPHAECR	Paramètre multiplicatif tunnel en écriture	AV^{-2}	68.6u
BETAECR	Paramètre exponentiel tunnel en écriture	Vm^{-1}	$2.62\ 10^{10}$
ALPHAEFF	Paramètre multiplicatif tunnel en effacement	AV^{-2}	16.6u
BETAEFF	Paramètre exponentiel tunnel en effacement	Vm^{-1}	$2.49\ 10^{10}$
VJ	Barrière de potentiel	V	1
MJ	Facteur de gradualité	-	0.5
TNOM	Température nominale	°	27.0

Tableau 3-3: Les paramètres du modèle et les paramètres liés à la composante de la cellule EEPROM.

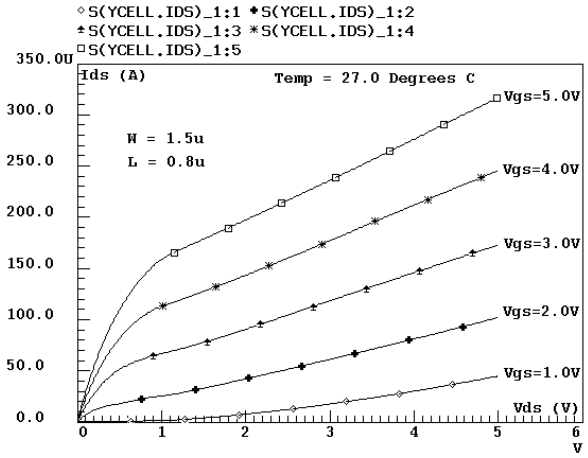
3.2.3 Implantation du modèle dans VHDL-AMS (voir annexe 2)

3.2.3.1 Régime statique

Dans ce régime, nous nous intéressons aux conditions de lecture de la cellule vierge. Dans ces conditions la charge de la grille flottante est nulle, c'est-à-dire **QFGO = 0**. Les conditions de lecture d'une cellule se font en général en régime linaire. Pour celà, nous simulons la caractéristique $I_{DS} = f(V_{CGS})$ tout en fixant la tension V_{DS} à 100mV et la valeur limite de la tension de grille de contrôle V_{cgs} entre 4 à 6V (Figure 3-11-a). La figure 3-11-b montre la caractéristique de sortie de la cellule ($I_{DS} = f(V_{DS})$)



a : caractéristiques $I_{DS} = F(Vcgs)$
La courbe neutre à $V_{BS} = 0V$



b : caractéristiques $I_{DS} = F(V_{ds})$

Figure 3-11 : Caractéristiques d'entrée et de sortie d'une cellule mémoire EEPROM en régime statique.

3.2.3.2 Régime transitoire

3.2.3.2.1 Opération d'effacement

Il est important, avant de montrer les résultats de simulation pour la validation du modèle, de donner les conditions de polarisation appliquées à la cellule durant l'opération d'effacement (Figure 3-12).

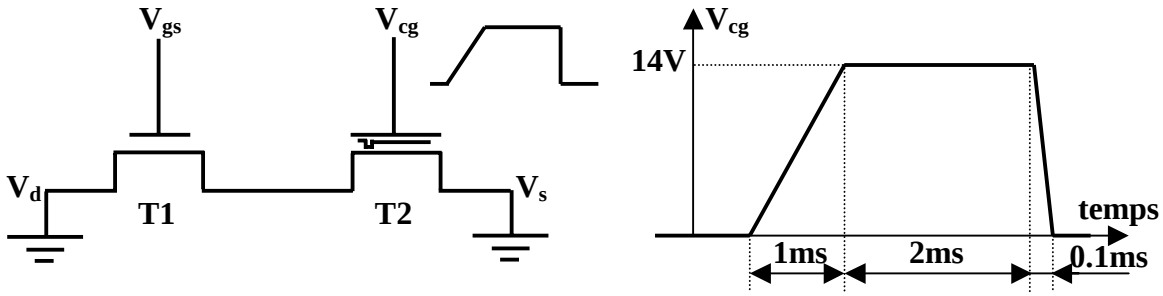


Figure 3-12 : Conditions d'effacement de la cellule EEROM.

Les tensions appliquées sont déterminées à partir de la technologie de l'EEPROM. Dans notre cas, nous utilisons une technologie simple polysilicium. Il faut regarder aussi la qualité et la tension de claquage de l'oxyde tunnel [BEN99]. Pour que nous favorisions le passage d'électrons du drain vers la grille flottante, nous appliquons une impulsion V_{gc} de l'ordre de 14V tout en maintenant le drain et la source à la masse. Ceci permet de créer un fort champ électrique qui favorise le passage des électrons (Figure 3-12).

Pour éviter l'apparition d'un courant drain, nous connectons la source du transistor d'état à la masse ($V_{DS} = 0$).

Nous présentons sur les figures 3-13 et 3-14 les principaux résultats de simulation du modèle du transistor d'état. La polarisation $V_{cg} = V(1)$ augmente la valeur du potentiel de la grille flottante V_{fg} et par conséquent celle du champ électrique tunnel E_{tun}. La diminution de la charge Q_{fg} ($Q_{fg} < 0$) est due à l'injection d'électrons dans la grille flottante. Cette diminution de Q_{fg} est à l'origine de la variation de la tension de seuil V_{tcg} du transistor d'état et la diminution de la tension de grille flottante V_{fg} quand la grille de contrôle V_{cg} atteint le palier ($V_{cg} = 14V$).

A la fin de l'impulsion de programmation (V_{cg}) de l'opération d'effacement, le potentiel V_{fg} tend vers sa valeur d'équilibre déterminée par le signe et la valeur de la charge Q_{fg} ($Q_{fg} < 0$).

Le potentiel de la grille flottante est couplé à la grille de contrôle V_{cg} par le coefficient K_{eff}. Ceci est valable lorsque la tension V_{cg} est non nulle. Le coefficient K_{eff} dépend de la capacité C_{PP} et les autres capacités de la structure (Equation 3.31). Nous remarquons que lorsque la grille de contrôle atteint le palier ($V_{cg} = 14V$), la capacité C_{PP} est en régime de désertion. Ceci provoque une diminution du coefficient de couplage en effacement K_{eff} ($K_{eff} = 0.64$). La nouvelle valeur de la capacité C_{PP} diminue l'efficacité de l'opération d'effacement.

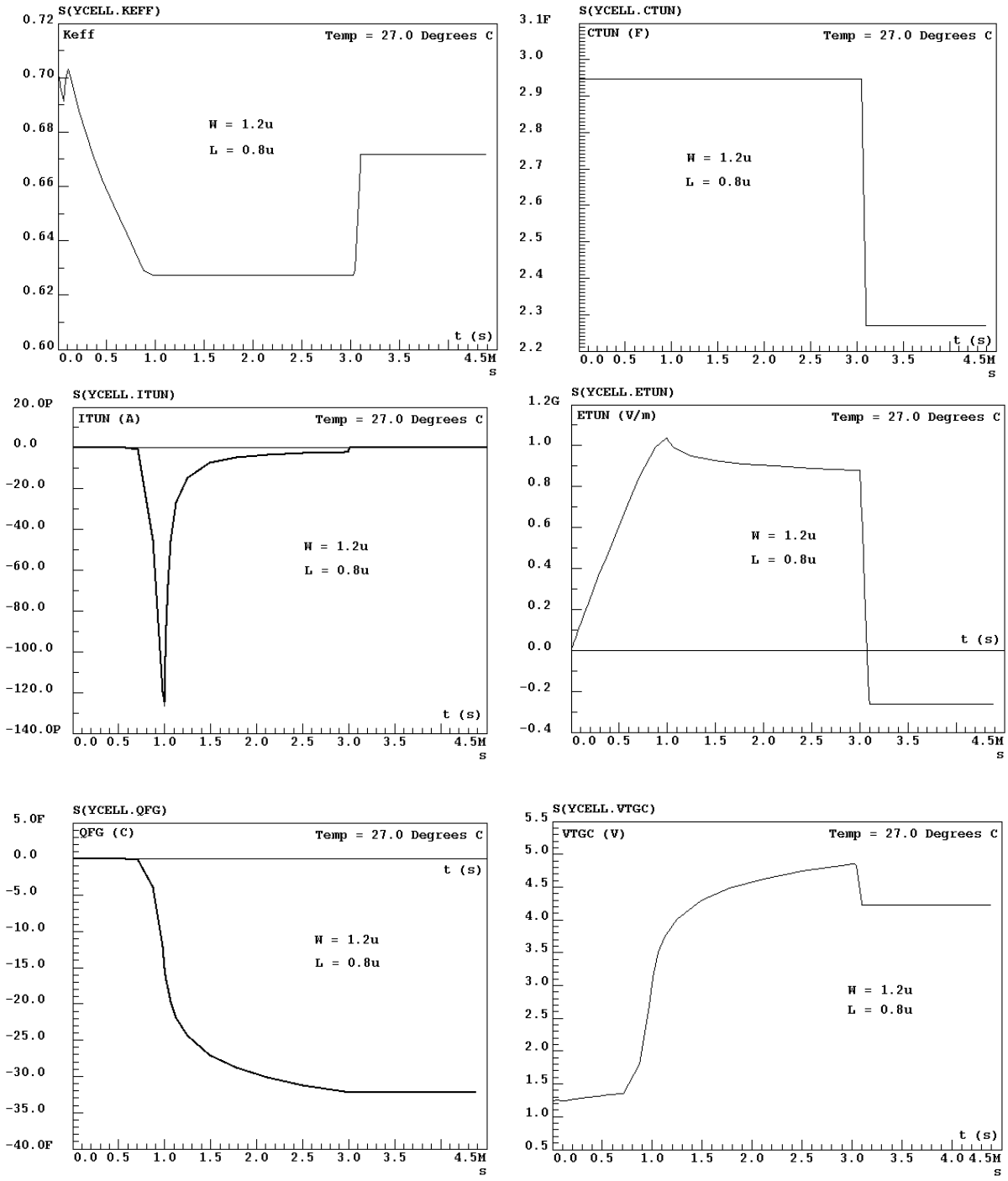


Figure 3-13 : Simulation de l'opération d'effacement.

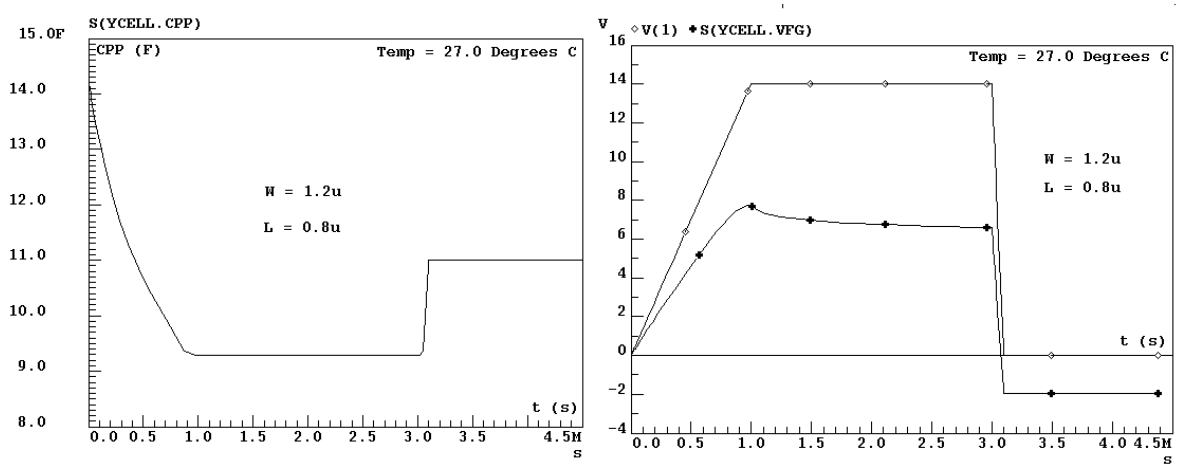


Figure 3-14 : Simulation de l'opération d'effacement.

La valeur de la tension de seuil de la cellule effacée (V_{TH} = 4.7V) est calculée à partir de la simulation en régime statique de la caractéristique $I_{DS} = f(V_{cgs})$ pour laquelle la valeur de la charge Q_{fg} = -32.21 fC (valeur obtenue après effacement) est remplacée dans le paramètre **QFGO** du modèle. La figure 3-15 montre le déplacement de la courbe $I_{DS} = f(V_{cgs})$ par rapport à la courbe neutre (Figure 3-11-a).

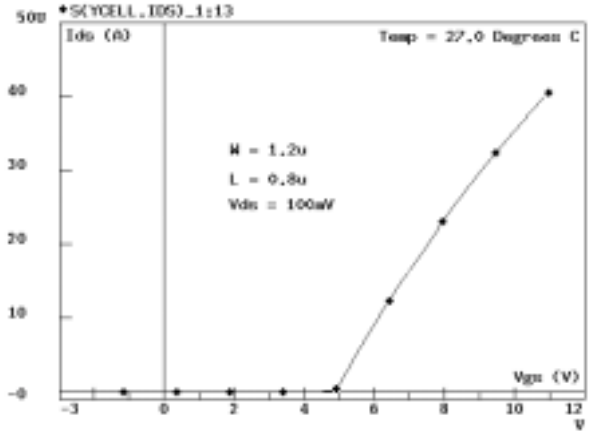


Figure 3-15 : Simulation de la caractéristique $I_{DS} = F(V_{gs})$ après effacement.

3.2.3.2.2 Opération d'écriture

L'opération d'écriture consiste à injecter des électrons depuis la grille flottante vers le drain par la création d'un champ électrique Etun à travers la zone tunnel. Ceci est réalisé par l'application d'une polarisation sur le drain de la cellule (Figure 3-16).

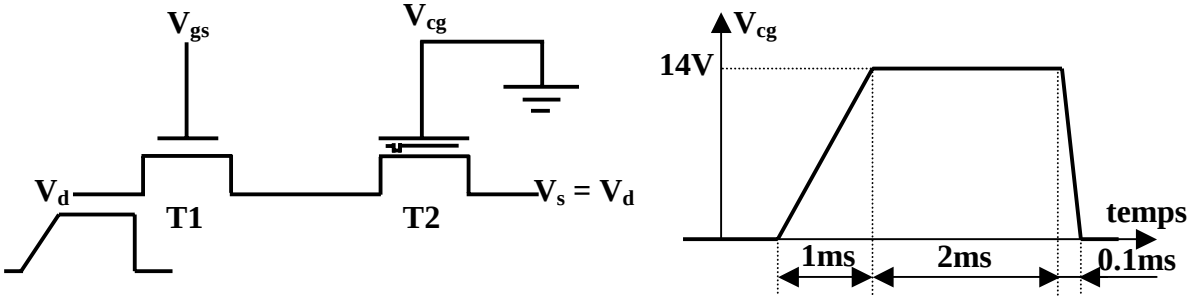


Figure 3-16 : Conditions d'écriture de la cellule EEROM.

Dans ce cas, c'est-à-dire l'opération d'écriture, le coefficient K_{ecr} dépend fortement de la capacité tunnel C_{tun}, mais également des autres capacités puisque K_{ecr} est le rapport de C_{tun} par la somme des toutes les capacités de la structure (Equation 3.31). La diminution de la capacité C_{tun} affectant la quantité de charges injectées dans le drain. Dans toute la durée de la programmation de la cellule, les capacités formées par la grille de contrôle et la grille flottante (C_{PP}) restent constantes. La variation de la tension de seuil V_{tgc} est due au coefficient K_{eff} qui varie principalement par la diminution de la capacité tunnel C_{tun} (Figure 3-17 et 3-18).

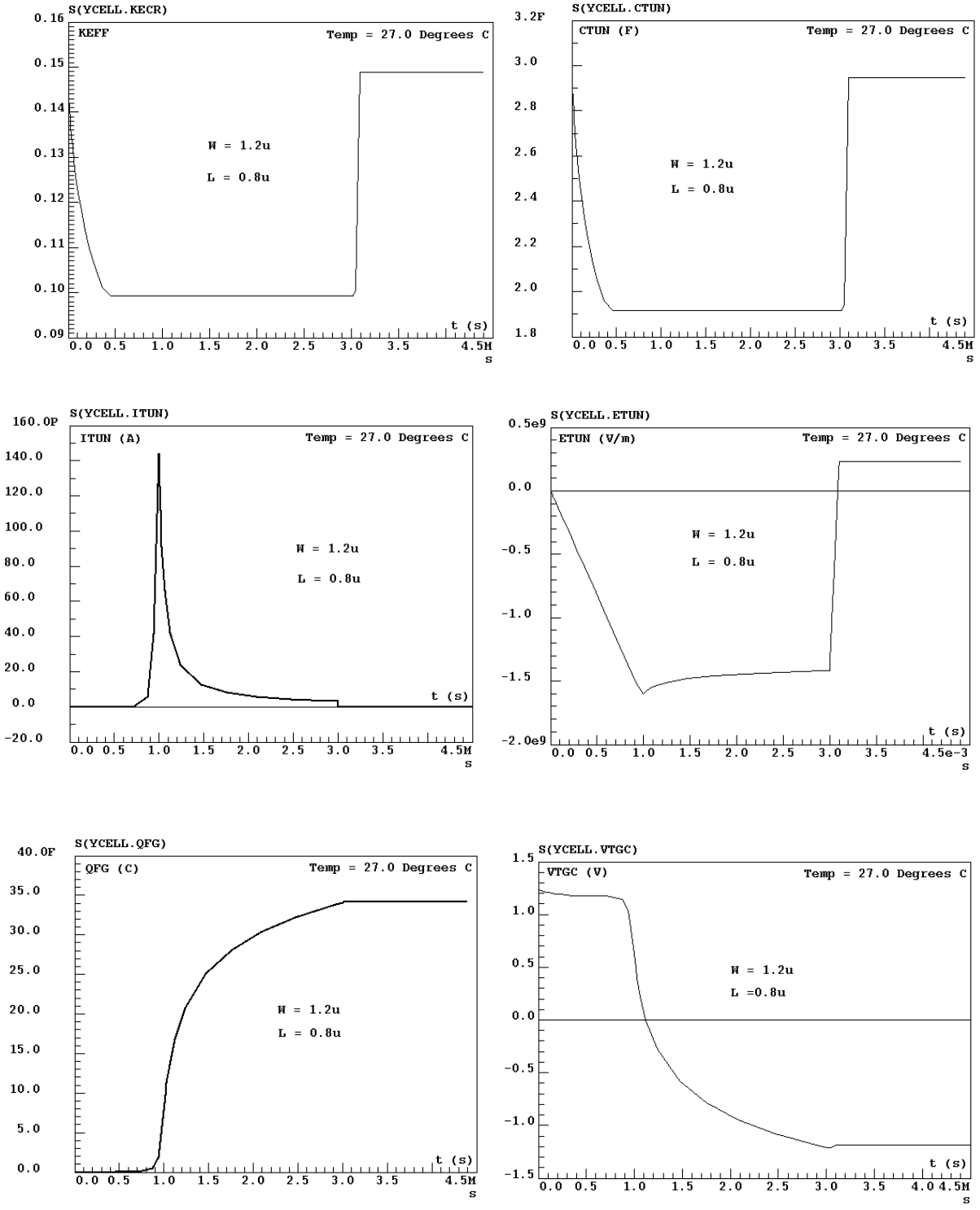


Figure 3-17 : Simulation de l'opération d'écriture.

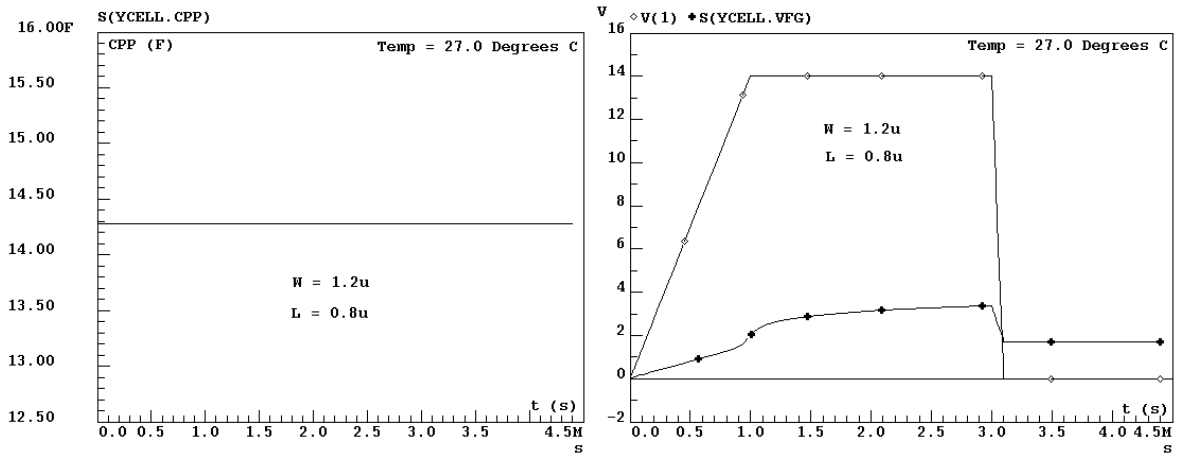


Figure 3-18 : Simulation de l'opération d'écriture.

La variation des résultats de simulation est effectuée comme dans le cas de l'opération d'effacement. La charge finale Q_{fg} = 34.12 fC, après écriture, est remplacée dans le paramètre **QFGO** du modèle. La figure 3-19 montre le déplacement de la courbe $I_{DS} = f(V_{cgs})$ par rapport à la courbe neutre (Figure 3-11-a).

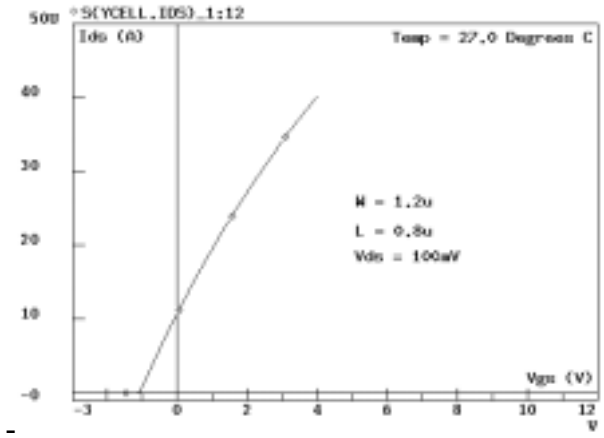


Figure 3-19 : Simulation de la caractéristique $I_{DS} = F(V_{gs})$ après effacement.

La figure 3-20 synthétise les résultats de simulation de la caractéristique statique de la cellule EEPROM vierge, après écriture et après effacement.

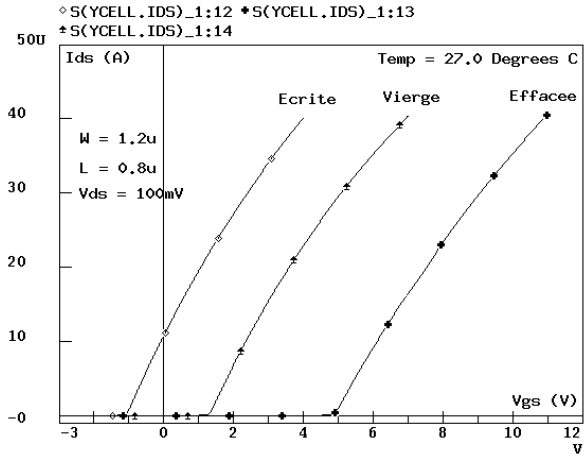


Figure 3-20 : Simulation de des caractéristiques $I_{DS} = F(V_{gs})$ vierge (neutre), après écriture et après effacement.

La figure 3-21 montre la simulation d'un point mémoire en utilisant en même temps les deux transistors : transistor d'état et transistor de sélection.

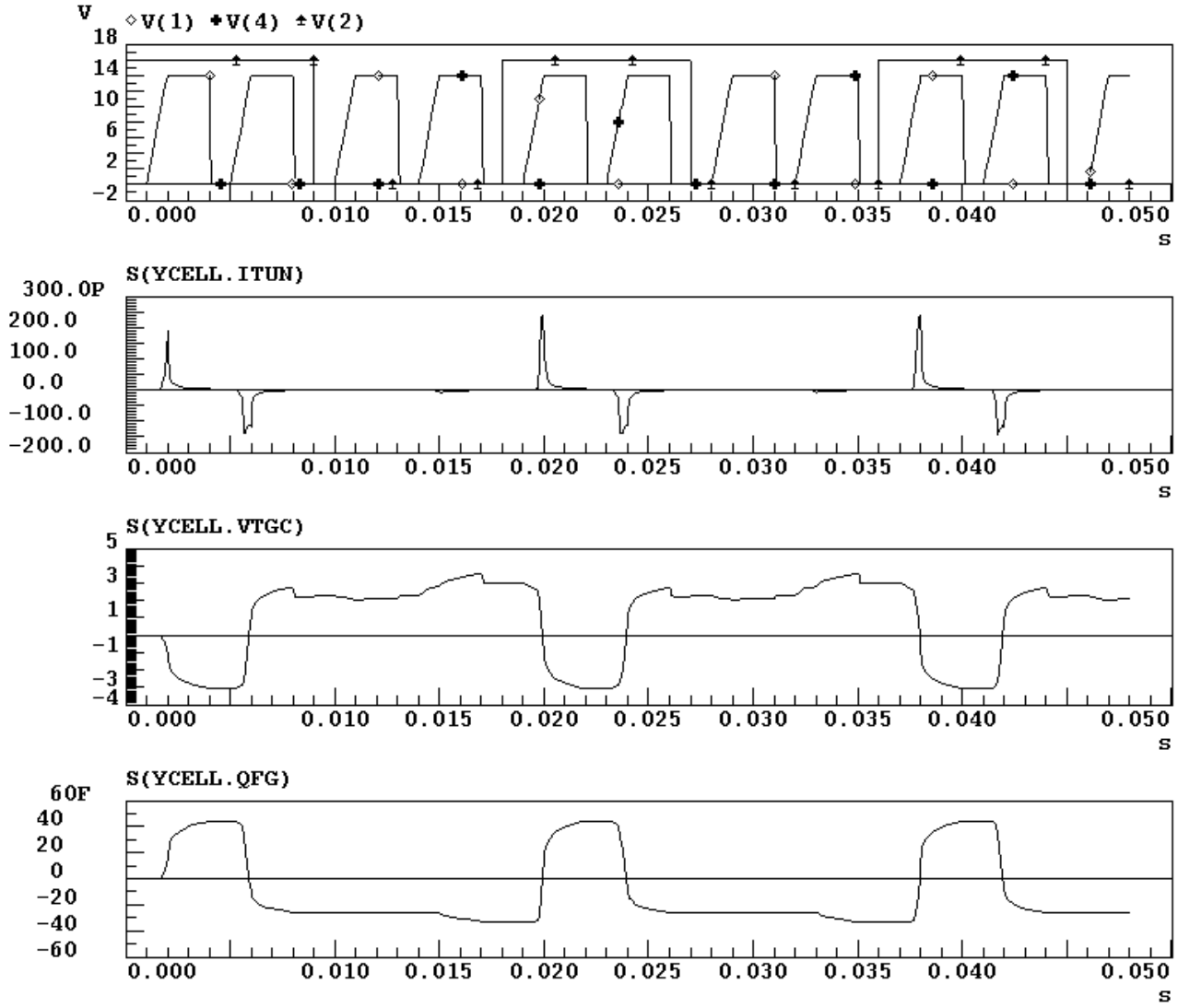


Figure 3-21 : Trois cycles d'écriture et d'effacement.

3.3 Modèle UNICELL du transistor MOS

Le modèle UNICELL est un modèle dérivé du Modèle à Charges Distribuées du GET/ENST décrit dans [JEM00] et écrit en VHDL-AMS dans l'annexe 3.

Nous présentons dans cette section les résultats de simulation avec le modèle UNICELL :

La figure 3-22 montre les caractéristiques de Id en fonction de Vd en faisant varier Vg. La figure 3-23 montre les caractéristiques de Log(Id) en fonction de Vg en faisant varier Vb. On constate sur la figure 3-23 que le courant sous-seuil de UNICELL est naturellement continu entre le régime de faible et de forte inversion.

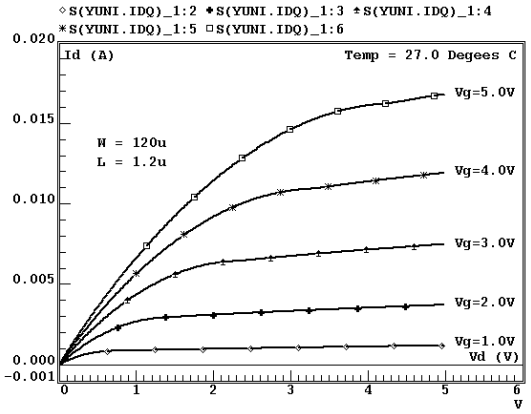


Figure 3-22 : Caractéristique Id = f(Vd).

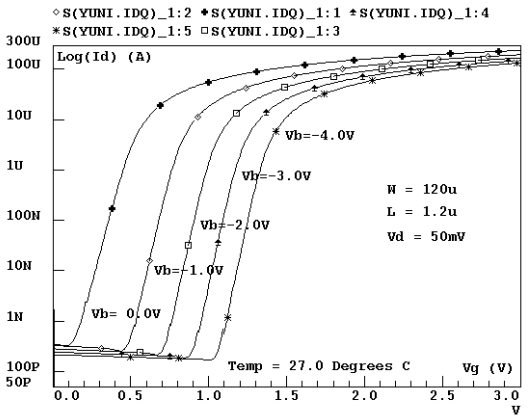


Figure 3-23 : Caractéristique Log(Id) = f(Vg).

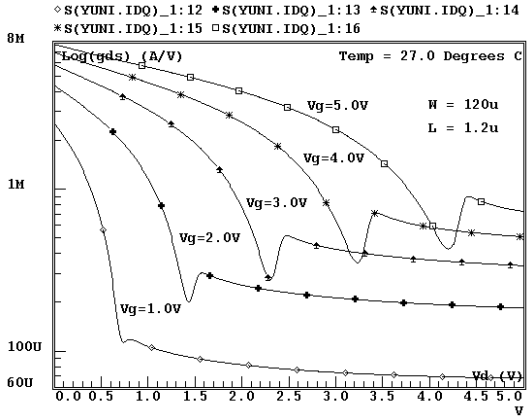


Figure 3-24 : Caractéristique Log(gds) = f(Vd).

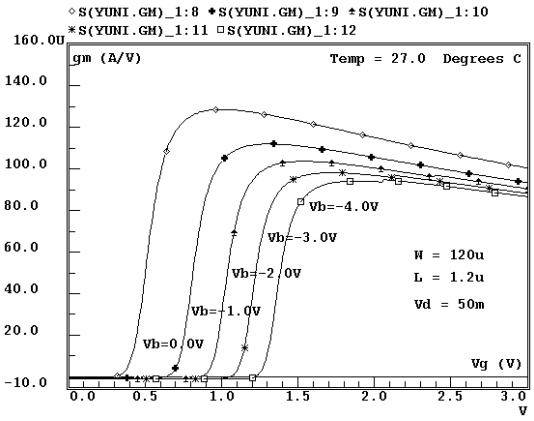


Figure 3-25 : Caractéristique gm = f(Vg).

Comparaison des Performances des Modèles MOS3 et UNICELL :

En vue de comparer les performances des modèles VHDL-AMS MOS3 de UNICELL nous avons choisi le transistor NMOS décrit dans [VLAD94]. Dans la figure 3-26 le courant sous-seuil est exponentiel ce qui caractérisent un courant de diffusion. Dans la transition faible-forte inversion le courant est continu naturellement avec UNICELL et artificiellement (par l'intermédiaire du paramètre NFS).

La figure 3-27 représente les caractéristiques gm (Vg). Le modèle MOS3 accuse un changement brusque de pente alors que la caractéristique de UNICELL est beaucoup plus « douce ». Il en est de même pour la caractéristique gds (Vd) de la figure 3-28.
UNICELL améliore la convergence de la simulation aux alentours du changement brusque de pente de ces caractéristiques.

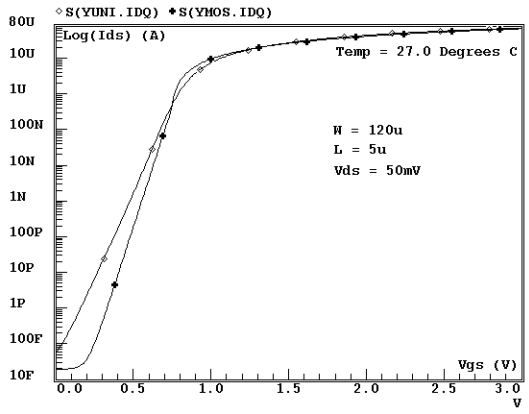


Figure 3-26 : Caractéristique Log(Ids) = f(Vgs).

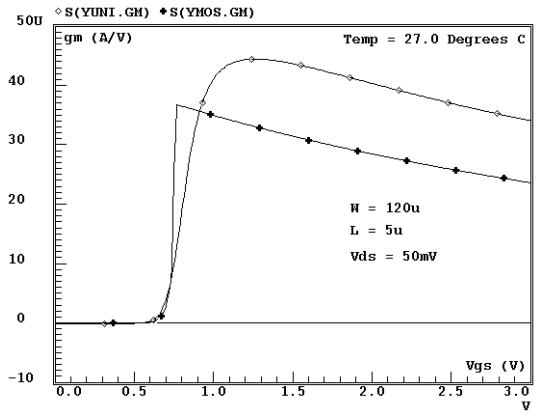


Figure 3-27 : Caractéristique gm = f(Vgs).

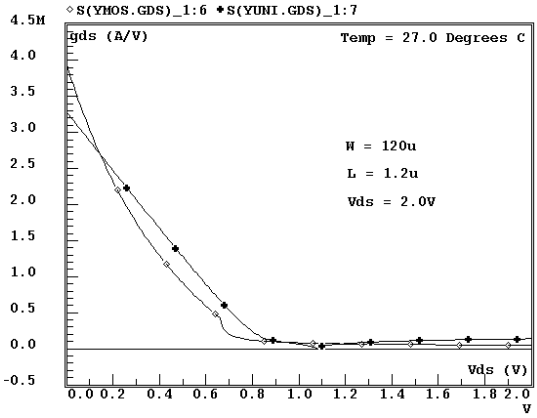


Figure 3-28 : Caractéristique gds = f(Vds).

3.4 Conclusion

Dans la première partie de ce chapitre, nous avons présenté les inconvénients ou les insuffisances apportées par les modèles de transistors MOS SPICE LEVEL 1 et 2 qui est rectifié par le SPICE niveau 3 et implanté ce dernier en DC et en TRAN. L'implantation est effectuée sur ADVance MS^{TM}[ADV99].

Dans la deuxième partie, nous avons présenté le modèle du transistor MOS à grille flottante qui s'agit d'une mémoire de type EEPROM et nous l'avons implanté en statique et en dynamique en utilisant un modèle de transistor MOS LEVEL 1 et un transistor MOS LEVEL 3 au niveau courant drain-source I_{DS}. Nous avons ensuite présenté les courbes des opérations d'écriture et d'effacement et leurs effets sur la caractéristique statique de $I_{DS} = f(V_{DS})$ avec le transistor MOS LEVEL 1.

Enfin, dans la dernière partie, nous avons comparé les performances des modèles UNICELL MOS LEVEL 3 en DC.

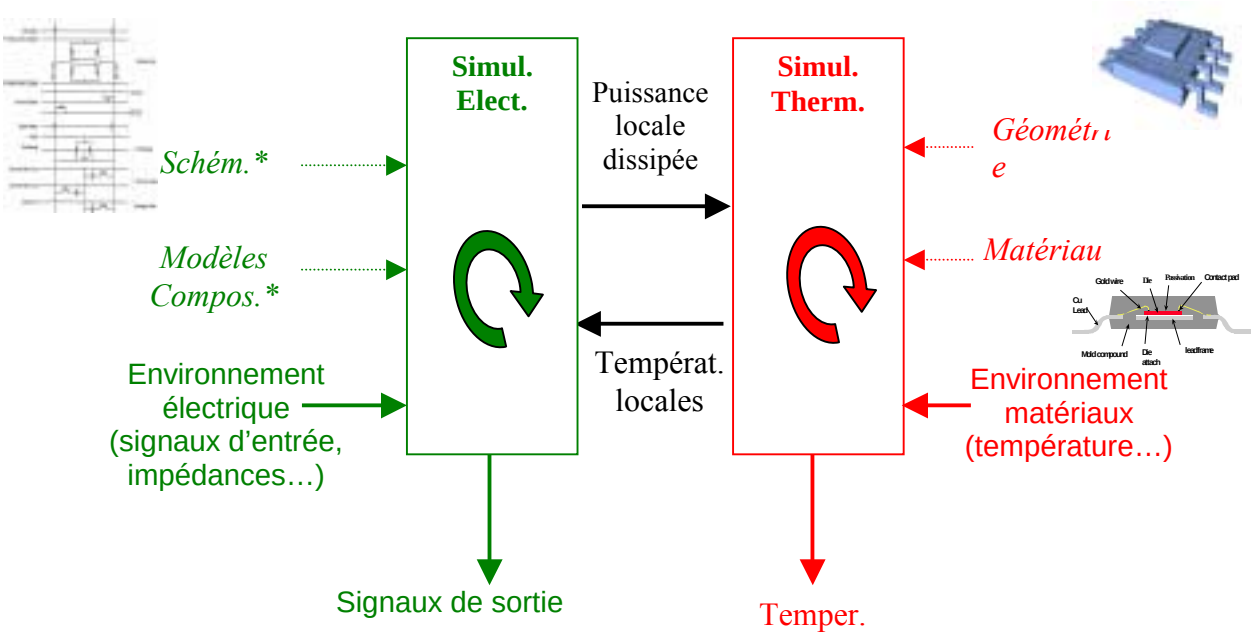
[2] Le modèle UNICELL de type modèle en feuillet (charge sheet model) de l'ENST qui est développé par le Prof. Jean-Jacques CHARLOT.

Dans cette étude nous avons adopté une approche ascendante qui consiste à commencer par développer un modèle simple, puis le compliquer petit à petit en incluant des équations plus complexes.

Nous concluons que le langage VHDL-AMS apporte souvent plus de souplesse que la modélisation ou la macro-modélisation de type SPICE. Cet apport, du langage VHDL-AMS, réside dans le fait que nous pouvons améliorer la convergence par la nouvelle instruction ***break***. Cette instruction élimine les discontinuités qui excitent dans le modèle. Avec ce langage, il est possible de réaliser une modélisation explicite ou implicite. Dans le premier exemple, c'est-à-dire, le modèle du MOS niveau 3 du SPICE, nous avons défini toutes les équations du modèle explicitement. Contrairement au deuxième exemple, une des équations du modèle de la cellule mémoire EEPROM est définie implicitement, c'est l'équation de la grille flottante. Avec SPICE, il est impossible de modéliser implicitement car nous utilisons que des composants et non pas des expressions.

4

MODELISATION - SIMULATION ELECTRO-THERMIQUE DES MEMOIRES FLASH ET SDRAM



Chapitre 4

4 MODELISATION – SIMULATION ELECTRO-THERMIQUE DES MEMOIRES FLASH ET SDRAM

4.1 Introduction

Ce chapitre est relatif à notre participation directe au projet européen SPARTE [SPA00]. Cette partie débute par une introduction sur la méthodologie générale de la simulation électro-thermique, méthodes directe et indirecte. De cette introduction est extraite la méthode utilisée dans le cadre de SPARTE : la méthode explicite à deux étapes qui sera appliquée aux mémoires FLASH et SDRAM prises comme démonstrateurs dans le cadre du projet. L'étude concernant la mémoire FLASH sera faite à l'aide de l'outil de simulation ADVanceMS de Mentor Graphics et celle relative à la mémoire SDRAM sera complètement menée avec VamSpiceDesigner. Cette dernière sera complétée par une étude comparative des résultats de mesure et de simulation en fonction de la température.

4.2 Méthodologie générale de la simulation électro-thermique

4.2.1 Introduction

La principale difficulté rencontrée dans la simulation électro-thermique est la complexité, du fait qu'une telle simulation doit gérer deux types d'algorithmes différents associés à deux types de phénomènes :

Simulation électrique :
- Variables électriques (tensions, courants, et paramètres de composants)
- Equations diverses très souvent complexes (modèles de transistors, de diodes..)

Simulation thermique :
- Variables thermiques (températures et flux de chaleur fonction de la position…) et paramètres (dues matériaux)
- Equations simples et uniformes la plus part du temps simples (loi de Fourier)

Les deux types de simulation sont complexes et leur couplage est plus complexe encore. Les flux de données sont synthétisés sur la figure 4-1.

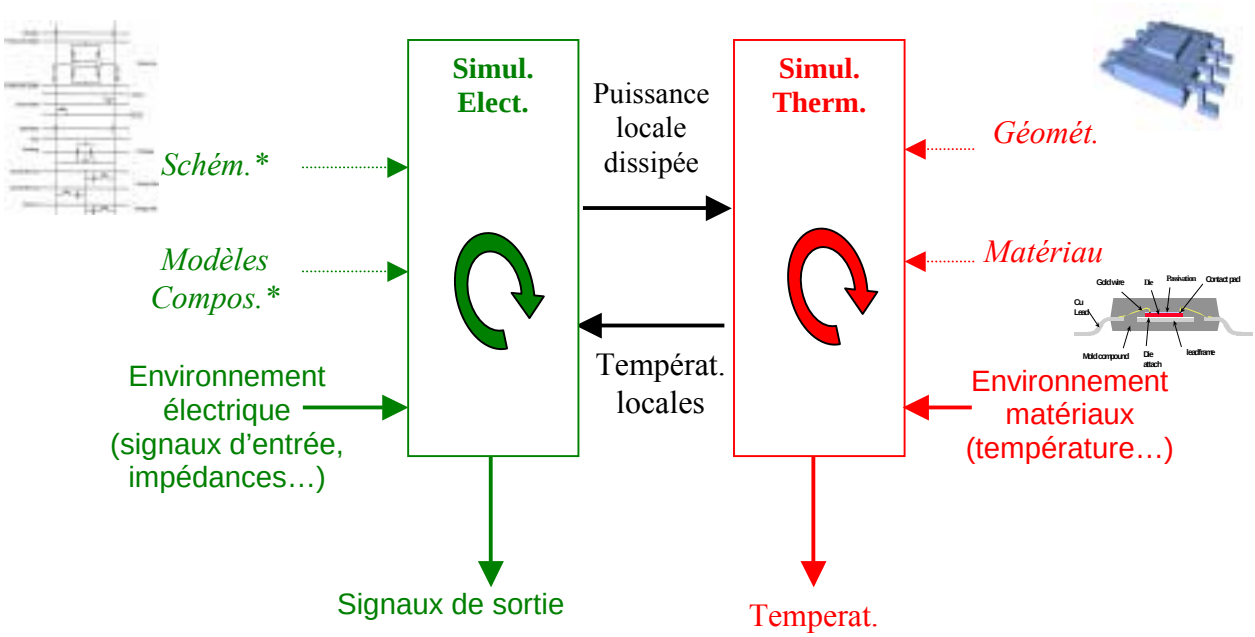


Figure 4-1 : Flot de données de la simulation électro-thermique.

Quelques flots de données* (dépendant du temps interviennent une fois seulement au début de la simulation)
D'autres flots de données interviennent durant tout le processus de simulation.

Deux méthodes électro-thermiques existent:

- ⇨ La 1ère est la méthode implicite qui utilise un seul outil pour simuler le comportement à la fois électrique et thermique du circuit.
- ⇨ La 2nde est explicite ou de relaxation dans laquelle les équations thermiques et électriques sont résolues séparément et alternativement. Cette méthode requiert un logiciel superviseur qui entretient un dialogue entre le programme thermique et le programme électrique.

4.2.2 Analyse de la méthode explicite brute

Dans la méthode explicite, la simulation nécessite de partager la simulation en une simulation électrique et une simulation thermique (Figure 4-2). Les liens entre les deux simulations sont données par :

- La puissance P_n dissipée par chaque éliment du circuit.
- La température T_n de chaque éliment du circuit.

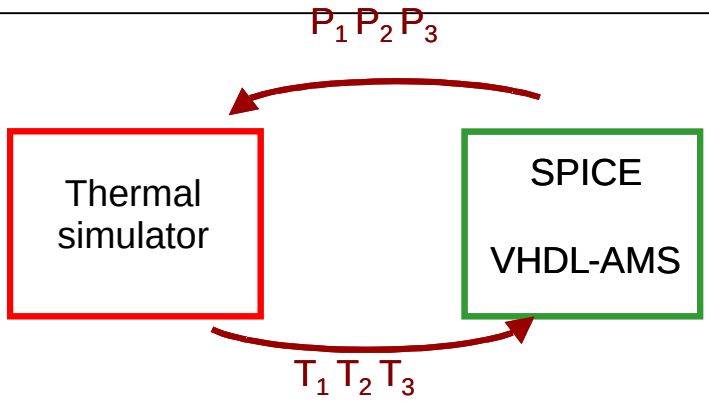


Figure 4-2 : Simulation électro-thermique avec la méthode explicite.

Théoriquement cette méthode devrait donner les résultats les plus précis par rapport aux autres méthodes déjà citées.

Le problème de cette approche réside dans la difficulté d'interconnecter les deux simulateurs. La conception du superviseur est complexe du fait qu'il doit gérer deux simulateurs qui n'ont pas été conçus pour échanger des données pendant leurs propres simulations, surtout quand les simulateurs sont localisés sur des systèmes informatiques différents et sur différents sites.

4.2.3 Variante de la méthode explicite : méthode en deux étapes

Une autre approche consiste à générer en utilisant un simulateur un modèle comportemental du système du point de vue du second simulateur (1ère étape), puis de simuler le système électro-thermique en utilisant le second simulateur et le modèle comportemental. Selon la nature du 1er simulateur, il y a alors deux possibilités :

La première possibilité (Figure 4-3) consiste à générer un modèle thermique comportemental à partir du simulateur thermique et de l'insérer dans le simulateur électrique. Le modèle comportemental thermique choisi est basé sur un réseau équivalent de résistances et de capacités thermiques qui est simulable facilement avec un simulateur électrique. Cette méthode conviendrait bien aux possibilités offertes par une description multi-technologique VHDL-AMS. Cependant, la génération de modèles comportementaux thermiques est très difficile dans la gamme de température –55°C à +125°C car le système est non-linéaire (i.e. les résistances et capacités dépendent des températures locales), problème que l'on ne sait pas résoudre à l'heure actuelle.

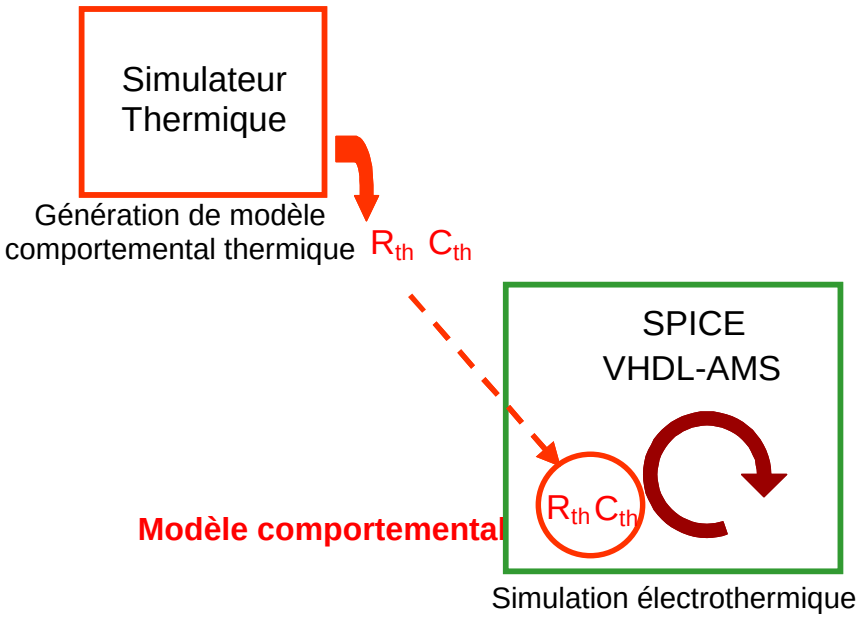


Figure 4-3 : Méthode en deux étapes avec un modèle comportemental thermique.

Symétriquement, il est possible de générer un modèle équivalent comportemental pour la partie électrique de la simulation et de l'injecter dans un simulateur thermique pour simuler les effets électo-thermiques (Figure 4-4). Le modèle comportemental électrique choisi est basé sur le calcul des puissances dissipées en fonction de la température pour chaque zone de dissipation. Nous pouvons diviser la simulation thermique en deux parties : simulation statique et transitoire.

- Statique
  P1(I0 or U0, I1 or U1,… Im or Um, T1,T2,…Tn),
  P2(I0 or U0, I1 or U1,… Im or Um, T1,T2,…Tn), …
  Pn(I0 or U0, I1 or U1,… Im or Um, T1,T2,…Tn),.

- Transitoire
  - **Première table**
    $I_0(t)$ or $U_0(t)$, $I_1(t)$ or $U_1(t)$,… $I_m(t)$ or $U_m(t)$
  - **Deuxième table**
    $P_1(I_0(t)$ or $U_0(t)$, $I_1(t)$ or $U_1(t)$,… $I_m(t)$ or $U_m(t)$, $T_1(t),T_2(t),\ldots T_n(t))$,
    $P_2(I_0(t)$ or $U_0(t)$, $I_1(t)$ or $U_1(t)$,… $I_m(t)$ or $U_m(t)$, $T_1(t),T_2(t),\ldots T_n(t))$, …
    $P_n(I_0(t)$ or $U_0(t)$, $I_1(t)$ or $U_1(t)$,… $I_m(t)$ or $U_m(t)$, $T_1(t),T_2(t),\ldots T_n(t))$,.

Ces fonctions doivent être déterminées par une simulation électrique. Le résultat de la simulation doit être fourni sous forme d'une table de valeurs pour les injecter dans un simulateur thermique.

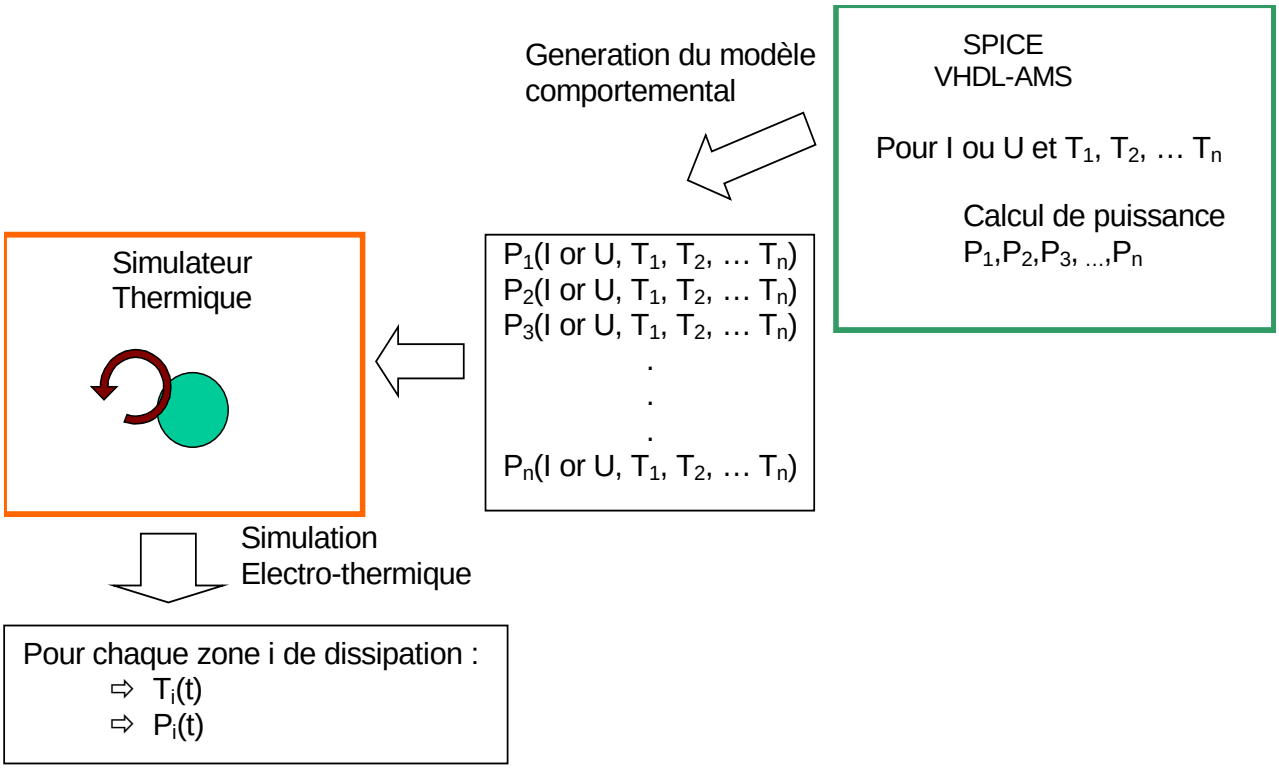


Figure 4-4 : Méthodologie de simulation électro-thermique.

C'est cette seconde méthode que nous avons retenue dans le cadre du projet SPARTE, et que nous allons appliquer successivement à deux types de mémoires – FLASH et SDRAM – qui sont deux des démonstrateurs du Projet.

4.3 Application de la méthode explicite à deux étapes à une mémoire FLASH

Nous avons appliqué la méthode explicite à deux étapes à la mémoire FLASH basée sur un réseau équivalent de résistances et capacités thermiques.

4.3.1 Modélisation d'une cellule de mémoire FLASH

Le système étudié comprend une cellule de mémoire FLASH et des circuits configurant la cellule dans le mode opératoire désiré, ici la lecture. La cellule de mémoire est constituée d'un transistor MOS et de deux diodes Fowler Nordheim qui interviennent dans le processus de charge ou de décharge de la grille flottante. La cellule est soudée sur une embase métallique (Figure 4-5) dont il faut considérer les effets thermiques à l'aide du réseau de résistances et capacités thermiques.

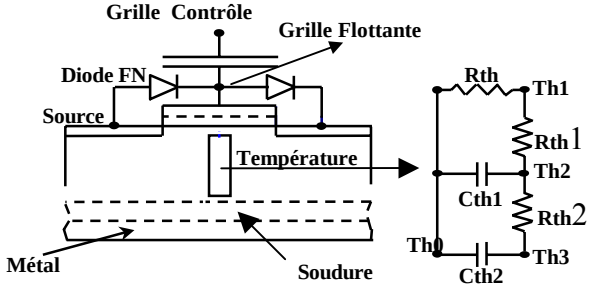


Figure 4-5 : Structure électro-thermique d'une cellule de mémoire Flash.

4.3.2 Analyse avec ADVanceMS de Mentor Graphics

Cet outil de simulation, entièrement compatible VHDL-AMS utilise le simulateur ELDO et le compilateur de chez LEDA [LED93]. La stratégie de simulation proposée correspond bien à une démarche hiérarchique, comme le montre la figure 4-7 extraite de la partie gauche de la figure 4-6. La structure indiquée reproduite la structure du fichier ***lectt1.cir*** et les différents *package* électrique (*elec*), thermique (*therm*) et radiatif (*rad*) définis dans le modèle de la cellule de la mémoire FLASH écrit en VHDL-AMS et occupant la partie droite de la figure 4-6.

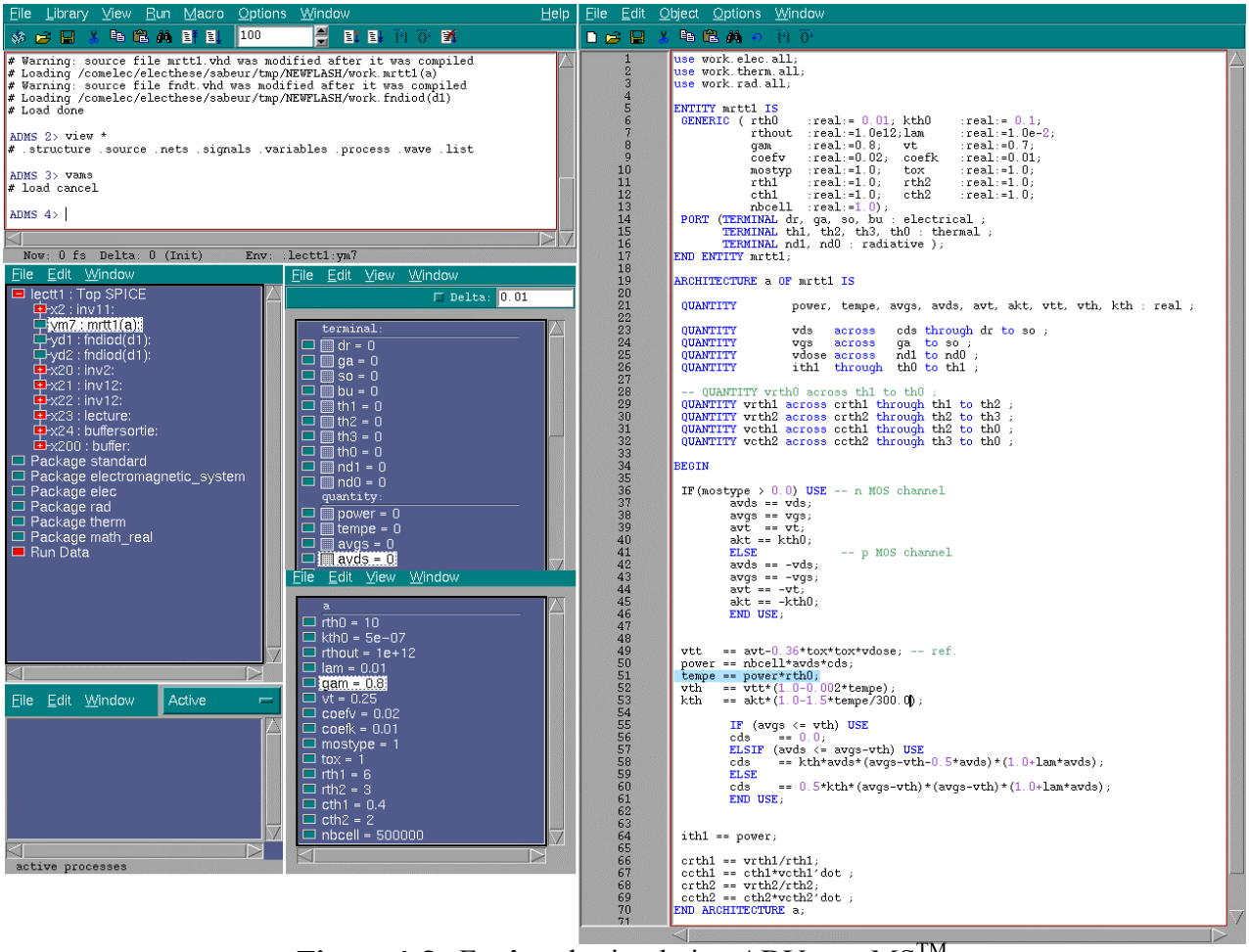


Figure 4-6 : Fenêtre de simulation ADVance MS™.

lectt1: Top SPICE
- lectt1:x2: inv11:
- lectt1:ym7: mrtt1(a):
- lectt1:yd1: fndiod(d1):
- lectt1:yd2: fndiod(d1):
- lectt1:x20: inv2:
- lectt1:x21: inv12:
- lectt1:x22: inv12:
- lectt1:x23: lecture:
- lectt1:x24: buffersortie:
- lectt1:x200: buffer:

std:standard: Package
disciplines:electromagnetic_system: Package
work:elec: Package

work:rad: Package
work:therm: Package
ieee:math_real: Package

Figure 4-7 : Structure de l'analyse ADVance MS^{TM}.

Les macro-composants (ou sous-circuits) SPICE du circuit de configuration électrique de la cellule sont indiqués par des termes de la forme **x**i, exemple **x23** pour le sous-circuit de *lecture.*
Les éléments modélisés en VHDL-AMS sont signalés par des termes de la forme **y**i :

ym7 => transistor MOS modélisé en VHDL-AMS par *mrtt1.vhd*
yd1 et yd2 => diode Fowler Nordheim modélisés par *fndt.vhd.*

Il est possible de « naviguer » dans cette structure en cliquant sur l'élément désiré et ainsi de modifier, par exemple, son modèle VHDL-AMS.
Cliquant sur **ym7** de la structure précédente, ADVance MS^{TM} fournit :

. une fenêtre appelée *variable* qui contient la liste des paramètres du modèle et leur valeur par défaut, figure 4-8 :

rth0 = 10	résistance thermique
kth0 = 5e-07	transconductance
rthout = 1e+12	résistance de sortie
lam = 0.01	modulation de long. de canal
gam = 0.8	facteur de seuil
vt = 0.25	tension de seuil
coefv = 0.02	coefficient Fowler Nordheim
coefk = 0.01	coefficient Fowler Nordheim
mostype = 1	type du MOSFET (ici, anal N)
tox = 1	épaisseur d'oxyde * 1.0-8
rth1 = 6	rés. thermique cellule 1
rth2 = 3	rés. thermique cellule 2
cth1 = 0.4	capa. thermique cellule 1
cth2 = 2	capa. thermique cellule 2
nbcell = 500000	nb de cellules de mémoire

Figure 4-8 : Paramètres du modèle.

Il est ainsi possible de modifier la valeur de ces paramètres dans une fenêtre appelée *nets* qui contient la liste des nœuds (***terminal***) et des variables (***quantity***) (et leurs valeurs après simulation) du modèle *mrtt1.vhd* correspondant dans la fenêtre variable de la figure 4-6.

Il est possible de sélectionner un (ou plusieurs) nœud(s) ou variable(s) et de le(s) charger pour effectuer une simulation et de visualiser les tensions aux nœuds ou la valeur des variables. En cliquant sur Top SPICE dans la structure, on fait apparaître dans la fenêtre *nets* la liste des nœuds du circuit de lecture. Nous avons sélectionner les nœuds *reade (grille de contrôle), th1, th2 et th3* pour visualiser les effets de l'auto-échauffement dans la mémoire (Figure 4-9).

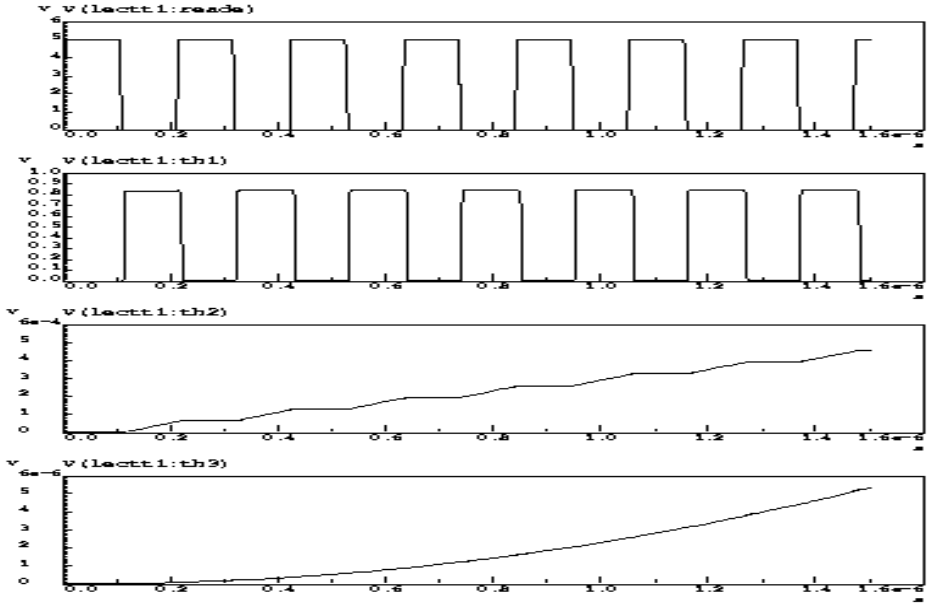


Figure 4-9 : Evolution des températures Th1, Th2 et Th 3 suivant un cycle de lecture.

Nous remarquons que l'effet de la température est faible, donc l'effet de l'auto-échauffement devient aussi faible. La puissance dissipée par la mémoire FLASH est très faible de l'ordre de ~50mW. Donc, l'étude de l'effet thermique pour évaluer les performances électriques de la mémoire FLASH est suffisante.

4.4 Modèle comportemental de calcul électro-thermique des mémoires FLASH et SDRAM à l'aide de la méthode explicites à deux étapes

Rappelons que les mémoires EPROM, FLASH, et SDRAM sont des mémoires non-volatiles comportant une structure très régulière. Elles sont organisées autour de matrices de cellules mémoires, chacune stockant un bit de donnée

Quelques études préliminaires ont montré que l'accroissement de température induit n'est pas très important (quelques degrés), ce qui laisse supposer que le couplage entre les effets électriques et thermiques est faible. Cette hypothèse réduit le nombre de simulations à un petit nombre de cycles

Le calcul des puissances dépend de l'application, et plus spécialement des séquences appliquées à l'entrée du circuit. Considérant que les mémoires étudiées comprennent des millions de cellules, une simulation complète au niveau transistor est impossible. De ce fait un modèle de calcul électro-thermique est rendu nécessaire.

Comme la plupart des composants logiques, la consommation des mémoires FLASH et SDRAM comprend deux parties :

- la consommation statique,
- la consommation transitoire.

La consommation totale est la somme de la consommation statique et transitoire. La puissance dissipée suit la même règle. De plus les cellules sont approximativement indépendantes. Les mémoires FLASH ou SDRAM possèdent un nombre très limité d'états : lecture, écriture, stand-by,

effacement partiel sélectif ou total. Ainsi, la puissance dissipée d'une cellule peut avoir 5 valeurs (une pour chaque état). Le nombre de types de transition est limité aussi du fait que certaines transitions sont interdites pour une bonne utilisation du circuit (par exemple, un effacement ne peut pas être suivi immédiatement d'une écriture). Ainsi, comme les mémoires considérées sont des composants MOS, l'énergie E_{total} pour une cellule durant un temps t pour une séquence donnée peut s'exprimer par :

$$E_{total} = \sum_{S} P_S \, \Delta t_S + \sum_{T} E_T \, n_T \tag{4.1}$$

Où :

S tous les états possibles
Ps est la puissance dissipée dans l'état S
Δt_S est la durée cumulée dans l'état S pour la séquence durant le temps t
T toutes les transitions possibles
E_T est l'énergie dissipée dans la transition T
n_T est le nombre de transitions T pour la séquence durant t

et

$$t = \sum_{S} \Delta t_S \tag{4.2}$$

Les paramètres de ce modèle sont P_S et E_T (pour chaque S et chaque T) : ils doivent être calculés par une simulation électrique en utilisant SPICE et/ou VHDL-AMS au niveau transistor d'une cellule mémoire, pour chaque état et pour chaque transition. En vue d'être applicable à chaque température comprise entre –55°C et +125°C, la simulation doit être répétée pour 4 à 5 températures dans la gamme de température considérée, ainsi P_S et E_T peuvent être interpolées pour des valeurs de températures intermédiaires.

Finalement, la puissance moyenne dissipée pendant t par une cellule est :

$$P_{average} = \frac{E_{total}}{t} \tag{4.3}$$

Le modèle de calcul électro-thermique a été implanté dans VamSpiceDesigner en utilisant les possibilités schématiques de l'outil.

4.5 Application au traitement électro-thermique d'une mémoire SDRAM à l'aide de la méthode explicite à deux étapes avec VamSpiceDesigner

En application de la méthode décrite précédemment et utilisant VamSpiceDesigner, la modélisation et la simulation d'une mémoire SDRAM MICRON 64 Meg a été menée en vue de déterminer les puissances et les énergies de transition et les puissances moyennes pour une séquence donnée.

4.5.1 Circuit équivalent de la mémoire SDRAM pour calculer les puissances et les énergies

Le schéma du circuit test pour calculer les puissances et les énergies est dessiné à l'aide de la schématique ELECTRIC de VamSpiceDesigner comme montré sur la figure 4-10.

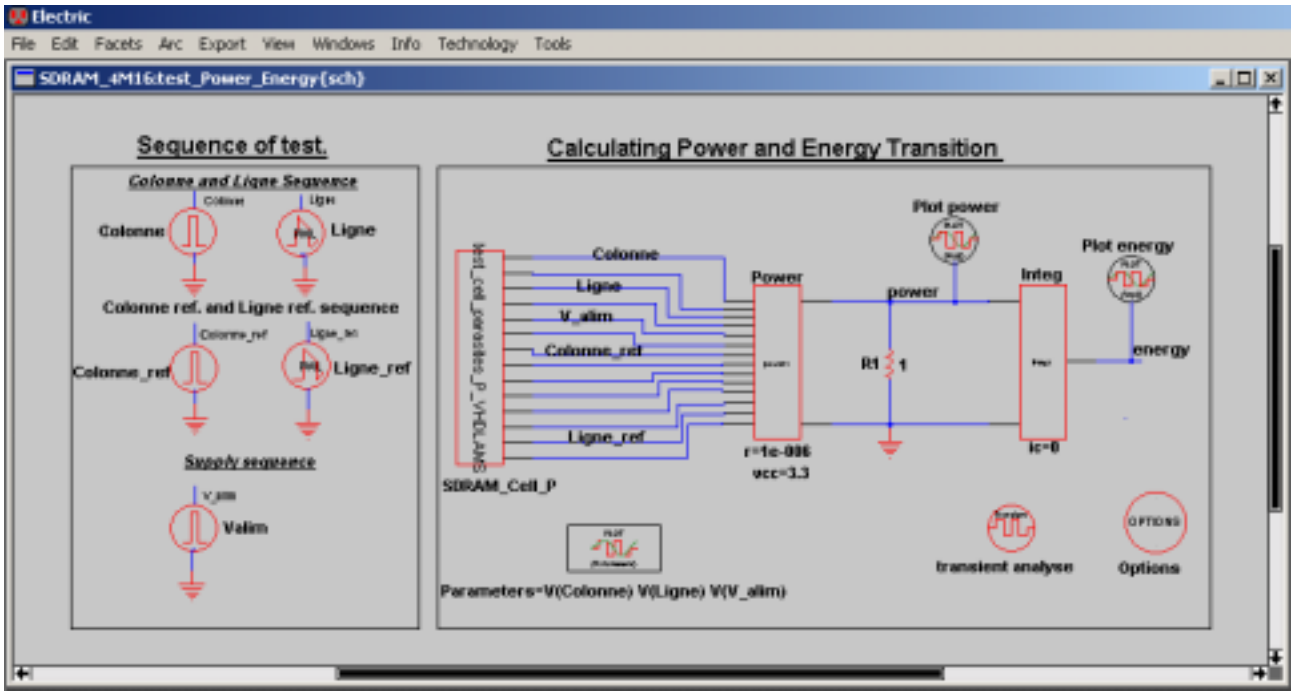


Figure 4-10 : Circuit équivalent de la mémoire SDRAM pour calculer les puissances, les énergies de transition et la puissance.

Trois rectangles apparaissent dans la partie droite de la figure 4-10 :

- Le rectangle de gauche contient le circuit équivalent de la cellule mémoire SDRAM (Figure 4-11), modélisé avec des modèles SPICE,
- Le rectangle du milieu (Power1) contient les éléments pour calculer les puissances instantanées. Sept paires de connections entre ce block et le block de gauche transfèrent les tensions aux bornes de très petites résistances (1μΩ) placées dans le circuit de la cellule mémoire pour recueillir la valeur du courant nécessaire au calcul de la puissance,
- Le rectangle de droite (Integ1) contient des éléments pour intégrer le signal de puissance instantanée issu du block de puissance pour obtenir les énergies de transition. Les cercles V représentent des appareils de mesure de la puissance et de l'énergie.

Comme mentionné dans le chapitre 1 on utilise les propriétés hiérarchiques de la schématique pour afficher le circuit lui même contenu dans le rectangle et en tapant *ctrl d* (d pour *down*).

De ce fait, on obtient le circuit de la cellule présenté sur la figure 4-11.

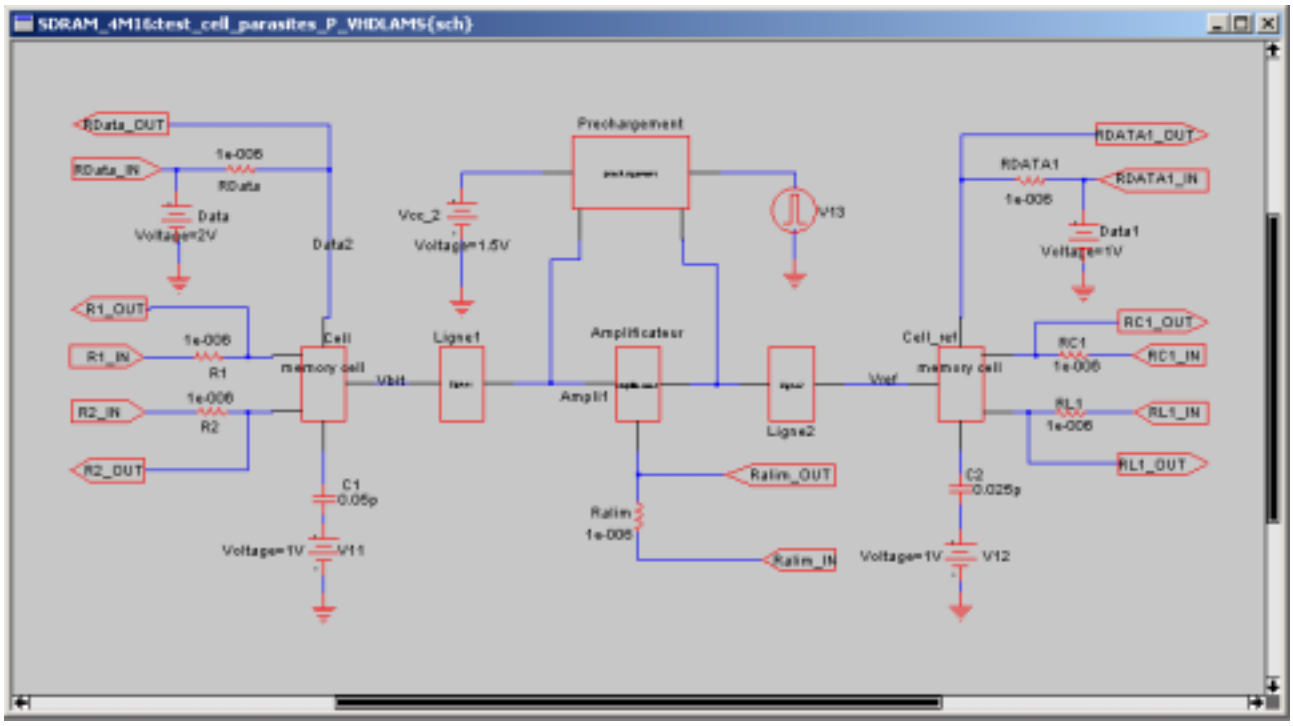


Figure 4-11 : Circuit de la cellule SDRAM avec ses sous-circuits.

4.5.2 Modélisation du circuit test

4.5.2.1 Modèle VHDL- AMS du block POWER1

La commande de la schématique ELECTRIC ***Make VHDL-AMS*** **View** dans le **Menu View** exercée dans la sous-boite POWER1 génère le squelette du modèle VHDL-AMS . Ce squelette doit être complété selon le désir de l'utilisateur, *en italique* dans la figure 4-12. Les *Quantities* V1 à V7 sont les tensions aux bornes *(across)* des résistances de 1µΩ. La puissance est obtenue en additionnant les valeurs absolues des tensions divisées par la valeur des résistances et multipliées par la tension d'alimentation VCC de la mémoire. La tension V3 est multipliée par 256, le nombre de cellules que nous avons considéré comme adressées par le processus d'amplification. La quantité POWER1 est traitée ici comme une quantité de branche *through quantity,* c'est pourquoi une résistance de 1Ω a été ajoutée entre la sortie du block et la masse pour obtenir en définitive l'équivalent d'une entrée *across* pour le block INTEG1.

```
-- VHDL-AMS automatically generated from facet power1{sch}
ENTITY power1 IS
 GENERIC(r : real:=1.0e-6;
        vcc : real := 3.3);
 PORT(TERMINAL out1, ref, in1, in2, in3, in4, in5, in6, in7, in8, in9,
 in10, in11, in12, in13, in14: Electrical);
END power1;
ARCHITECTURE power1_BODY OF power1 IS
  quantity V1 across in1 to in2;
  quantity V2 across in3 to in4;
  quantity V3 across in5 to in6;
  quantity V4 across in7 to in8;
```

```
  quantity V5 across in9 to in10;
  quantity V6 across in11 to in12;
  quantity V7 across in13 to in14;
  quantity power through ref to out1;
BEGIN
  power == ( (abs(V1) + abs(V2) +256.0*abs(V3) + abs(V4) + abs(V5) + abs(V6)  + abs(V7) ) / r)
* vcc;
END power1_BODY;
```

Figure 4-12 : Modèle VHDL-AMS du block Power1

4.5.2.2 Modèle VHDL- AMS du block INTEG1

Le modèle du block INTEG1 est généré automatiquement et complété (en *italique*) de la même façon que précédemment comme présenté dans la figure 4-13.

```
-- VHDL-AMS automatically generated from facet integ1{sch}
ENTITY integ1 IS
 GENERIC(ic :real:= 0.0);
 PORT(TERMINAL in1, ref: Electrical; QUANTITY out1: OUT Real);
END integ1;
ARCHITECTURE integ1_BODY OF integ1 IS
  quantity vin across in1 to ref;
BEGIN
  out1 == vin'integ;
END integ1_BODY;
```

Figure 4-13 : Modèle VHDL-AMS pour calculer les énergies de la cellule mémoire SDRAM.

4.5.2.3 Modélisation SPICE des transistors MOS et composants de base

Le circuit de la figure 4-11 contient d'autres boites qui peuvent être affichées à l'écran par voie hiérarchique. Ce circuit est totalement modélisé avec des modèles SPICE et le modèle MOS de niveau 3 de façon à prendre en compte les effets de la température.

4.5.3 Simulation du circuit test Puissance & Energie

4.5.3.1 Netlist

Une NETLIST peut être obtenue automatiquement en utilisant la commande *Simulation (SPICE)* → *Simulate* → *SPICE OPUS* (Figure 4-14).

```
*** FACET test_Power_Energy FROM LIBRARY SDRAM_4M16 ***
*** FACET CREATED Mon Jul 01 17:11:39 2002
*** VERSION 1 LAST REVISED Tue Aug 27 11:58:46 2002
*** EXTRACTED BY ELECTRIC DESIGN SYSTEM, VERSION 6.04
*** UC SPICE *** , MIN_RESIST 0.000000, MIN_CAPAC 0.000000FF
.model MODNMOS NMOS (
+ LEVEL = 3      VTO  = 0.8
+ CGSO  = 3E-10  CGDO = 3E-10 CGBO = 2E-9
+ RSH   = 300    CJ   = 1e-3  CJSW = 3.4E-10 JS    = 1E-7
```

```
+ TOX   = 80E-10 NSUB = 9E17  NFS  = 3e11    XJ    = 1E-7
+ LD    = 3E-8   UO   = 400   VMAX = 1.4E5   DELTA = 0.4    THETA = 0.06
+ ETA   = 2E-2   KAPPA= 0.055)

.model MODPMOS PMOS (
+ LEVEL = 3        VTO  = -0.8
+ CGSO  = 2.7E-10  CGDO = 2.7E-10 CGBO = 1.05E-9
+ RSH   = 700      CJ   = 2e-3    CJSW = 3.5E-10 JS    = 8E-7
+ TOX   = 80E-10   NSUB = 5E17    NFS  = 2e11    XJ    = 1E-7
+ LD    = 2E-9     UO   = 120     VMAX = 2.5E5   DELTA = 0.7    THETA = 0.13
+ ETA   = 3E-2     KAPPA= 2)

.SUBCKT integ1 in1 ref n1vcvsout1
ainteg1 in1 ref n1vcvsout1 modinteg1
.model modinteg1 integ1
+ ic = 0.0

.ENDS integ1

.SUBCKT cellule_memoire_parasite Data Colonne Ligne Bit_Ligne Vmemoire
** GROUND NET: 0 (net3)
** PORT Data
** PORT Colonne
** PORT Ligne
** PORT Bit_Ligne
** PORT Vmemoire
CC1 Bit_Ligne Ligne 10F
CC2 0 Bit_Ligne 500F
MM1 Bit_Ligne Ligne Vmemoire 0 MODNMOS L=0.18U W=0.6U
MM2 Data Colonne Bit_Ligne 0 MODNMOS L=0.18U W=0.6U
.ENDS cellule_memoire_parasite

.SUBCKT prechargement Out_Mem Out_Ref Vprecharge Vcc_2
** GROUND NET: 0 (net1)
** PORT Out_Mem
** PORT Out_Ref
** PORT Vprecharge
** PORT Vcc_2
Mnode6 Out_Ref Vprecharge Out_Mem 0 MODNMOS L=0.18U W=0.6U
Mnode7 Vcc_2 Vprecharge Out_Ref 0 MODNMOS L=0.18U W=0.6U
Mnode8 Vcc_2 Vprecharge Out_Mem 0 MODNMOS L=0.18U W=0.6U
.ENDS prechargement

.SUBCKT Ligne2 L_OUT L_IN
** GROUND NET: 0 (net1)
** PORT L_OUT
** PORT L_IN
RR1 L_OUT L_IN 100
CC1 0 L_OUT 1000F
.ENDS Ligne2
```

```
.SUBCKT Ligne1 L_OUT L_IN
** GROUND NET: 0 (net1)
** PORT L_OUT
** PORT L_IN
CC1 0 L_OUT 2.69P
RR1 L_OUT L_IN 700
.ENDS Ligne1

.SUBCKT amplificateur In_Amp Vref V_alim
** GROUND NET: 0 (net6)
** PORT In_Amp
** PORT Vref
** PORT V_alim
MM2 In_Amp Vref 0 0 MODNMOS L=0.18U W=0.6U
MM3 Vref In_Amp 0 0 MODNMOS L=0.18U W=0.6U
MM4 Vref In_Amp V_alim V_alim MODPMOS L=0.25U W=5.5U
MM1 In_Amp Vref V_alim V_alim MODPMOS L=0.25U W=5.5U
.ENDS amplificateur

.SUBCKT test_cell_parasites_P_VHDLAMS Colonne R1_OUT Ligne R2_OUT Colonne1
+  Ralim_IN Data11 Data Data1 Data2 RC1_OUT Ligne1 RL1_OUT Ralim_OUT
** GROUND NET: 0 (net36)
** PORT Colonne
** PORT R1_OUT
** PORT Ligne
** PORT R2_OUT
** PORT Colonne1
** PORT Ralim_IN
** PORT Data11
** PORT Data
** PORT Data1
** PORT Data2
** PORT RC1_OUT
** PORT Ligne1
** PORT RL1_OUT
** PORT Ralim_OUT
RRData Data2 Data 1e-006
RRDATA1 Data1 Data11 1e-006
RRalim Ralim_IN Ralim_OUT 1e-006
RRL1 Ligne1 RL1_OUT 1e-006
RRC1 Colonne1 RC1_OUT 1e-006
RR2 R2_OUT Ligne 1e-006
RR1 R1_OUT Colonne 1e-006
CC2 net37 Vmemoire1 25F
Xnode29 Data11 RC1_OUT RL1_OUT Vref Vmemoire1 cellule_memoire_parasite
CC1 net43 Vmemoire 50F
XCM1 Data2 R1_OUT R2_OUT Vbit Vmemoire cellule_memoire_parasite
Xnode25 net34 net33 net32 net30 prechargement
Xnode26 net33 Vref Ligne2
```

```
Xnode27 net34 Vbit Ligne1
XAmpli1 net34 net33 Ralim_OUT amplificateur
** Sources and special nodes:
VV12 net37 0 DC 1
VV11 net43 0 DC 1
VVcc_2 net30 0 DC 1.5
VV13 net32 0 PULSE 0 2 59990N 59991N 59999N 60M 60M
VValim Ralim_IN 0 DC 1 PULSE(0 3  60n 2n 2n 7n 40n)
*VValim Ralim_IN 0 DC 1 PULSE(0 3  63n 2n 2n 10n 60n)
*Vligne Ligne 0 DC 1 pulse (0 1.8 5n 2n 2n 6n 20n)
Vligne Ligne 0 DC 1 pwl(0 0 2n 1.8 8n 1.8 10n 0 60n 0)
*VColonne Colonne 0 DC 1 pulse (0 1.8 6n 2n 2n 5n 20n)
VColonne Colonne 0 DC 1 pulse (0 1.8 50n 2n 2n 5n 20n)
VData Data 0 DC 2V
VData1 Data1 0 DC 1V
Vcolonne_1 Colonne1 0 DC 1 pulse (0 1.8 50n 2n 2n 5n 20n)
VLigne_1 Ligne1 0 DC 1 pulse (0 1.8 5n 2n 2n 6n 20n)
.ENDS test_cell_parasites_P_VHDLAMS
.SUBCKT power1 out1 ref in1 in2 in3 in4 in5 in6 in7 in8 in9 in10 in11 in12 in13 in14
apower1 out1 ref in1 in2 in3 in4 in5 in6 in7 in8 in9 in10 in11 in12 in13 in14 modpower1
.model modpower1 power1
+ r = 1.0e-6
+ vcc = 3.3

.ENDS power1


*** TOP LEVEL FACET: test_Power_Energy{sch}
** GROUND NET: 0 (net20)
XInteg power 0 energie integ1
XCell_P net18 net17 net16 net15 net12 net14 net7 net10 net8 net19 net11 net6
+  net5 net13 test_cell_parasites_P_VHDLAMS
XPower power 0 net18 net17 net16 net15 net14 net13 net12 net11 net10 net19
+  net8 net7 net6 net5 power1
RR1 0 power 1

** Options and control:
.option temp = 125 reltol = 0.09
.control
 run
  tran 0.1n 450.5n 0.1n
  plot net18 net16 net14
  plot V(power)
  plot V(energie)
  let MaxTime = max(time)
  let MaxEnergie = max(energie)
  let MaxPower = max(power)
  let MaxCurrent = MaxPower/3.3
  let ap = MaxEnergie / MaxTime
```

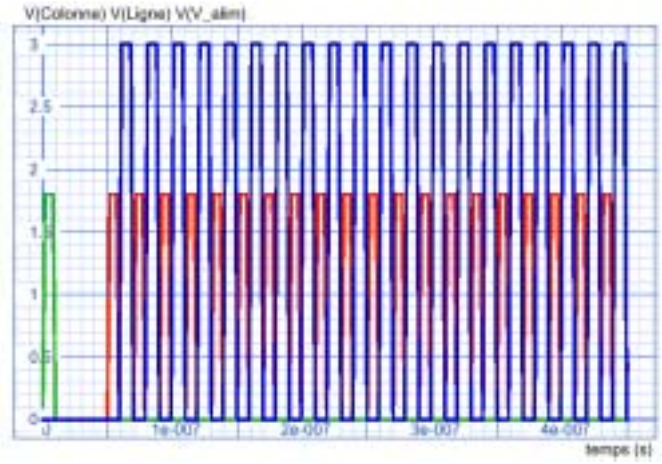
```
 print MaxTime
 print MaxEnergie
 print ap MaxPower
 print MaxCurrent

.ENDC
.END
```

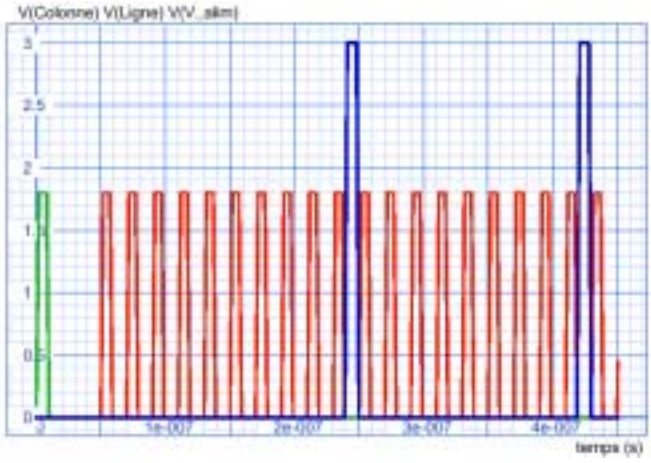
Figure 4-14 : NETLIST SPICE générée automatiquement par la schématique.

4.5.3.2 Séquence de test

La séquence de test choisie pour notre étude consiste en une écriture avec précharge suivie, après deux périodes de 10 Ns, par des séquences périodiques de 10 lectures et une amplification sur 256 éléments, comme présenté sur la figure 4-15.



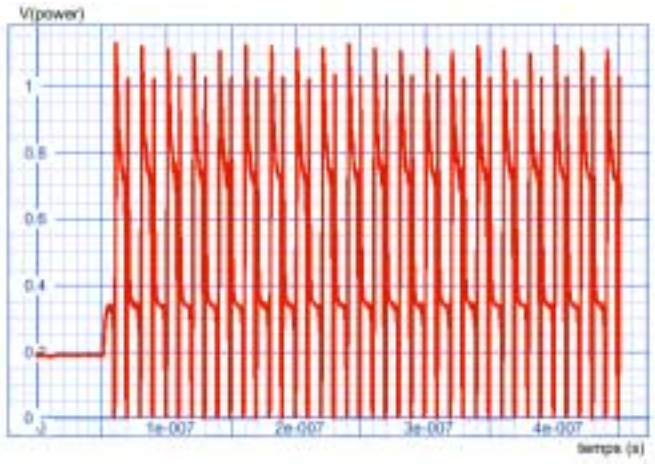
a) Amplification après chaque lecture
En ordonnée, l'axe des tensions en Volts, en abscisse le temps en secondes



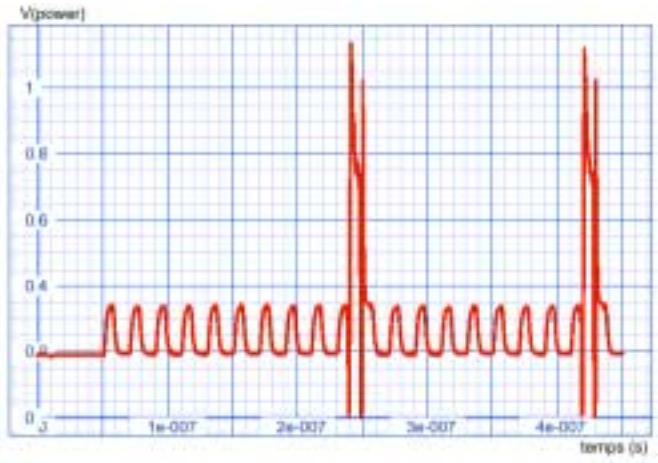
b) Amplification après 10 lectures

Figure 4-15 : Séquence de test.

4.5.3.3 Puissance instantanée à 25°C (Figure 4-16)



a) Amplification après chaque lecture
En ordonnée, l'axe courant en Ampères, en abscisse le temps en secondes



b) Amplification après 10 lectures
En ordonnée, l'axe des courants en Ampères, en abscisse le temps en secondes

Figure 4-16 : Courant instantané.

4.5.3.4 Energie de transition instantanée à 25°C (Figure 4-17)

a) Amplification après chaque lecture
En ordonnée, l'axe des énergies en Joules, en abscisse le temps en secondes

b) Amplification après 10 lectures
En ordonnée, l'axe des énergies en Joules, en abscisse le temps en secondes

Figure 4-17 : Energie de Transition en fonction du temps.

4.5.4 Comparaison des résultats de mesure et de simulation en fonction de la température

4.5.4.1 Mesure

Les tableaux suivants et les courbes correspondantes présentent les résultats en fonction de la température :

- Des courants mesurés,
- Des puissances calculées (= courants mesurés * VCC),
- Les coefficients de variation avec la température,
- Dans trois modes : opératoire *(operating)*, stand-by, et auto-rafraîchissement (*auto-refresh*)

Operating Mode			
Temperature	idd4_vcc3v6	Power	Variation coeff.
-55,1	62,89	226,404	
-45	62,37	224,532	-0,00827
25	59,5	214,200	-0,04602
85	57,42	206,712	-0,03496
125	55,99	201,564	-0,02490
150	54,95	197,82	-0,01857
Average Variation Coefficient			-2,654%

Tableau 4-1 : mode opératoire.

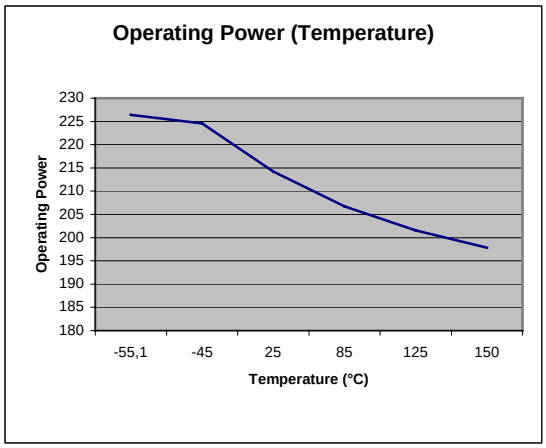


Figure 4-18 : Puissance Mode Opératoire

Temperature	idd2_vcc3v6	Power
-55	0,2228	0,80208
-46	0,2162	0,77832
25,3	0,1783	0,64188
85	0,2017	0,72612
125	0,3937	1,41732
150	0,8036	2,89296

Tableau 4-2 : Mode Standby.

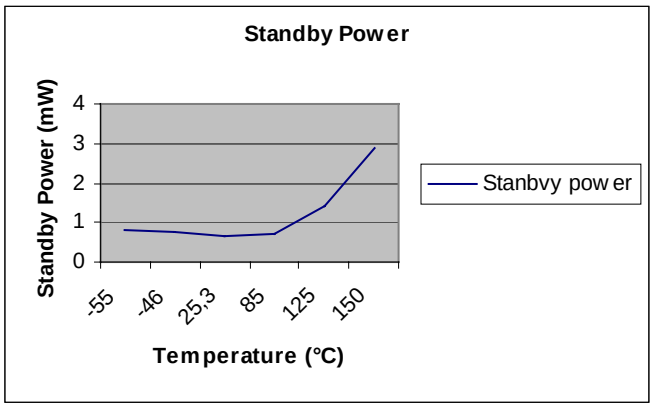


Figure 4-19 : Mode Standby.

Refresh Mode every 4096 Cycles			
Temperature	idd5_vcc3v6	Power	Variation Coeff.
-55	127,2	457,92	
-45	125,4	451,44	-0,01415
25	113,9	410,04	-0,09171
85	111,3	400,68	-0,02283
125	107,6	387,36	-0,03324
150	103,3	371,88	-0,03996
Average variation coefficient			-4,038%

Tableau 4-3 : Mode Auto-refresh.

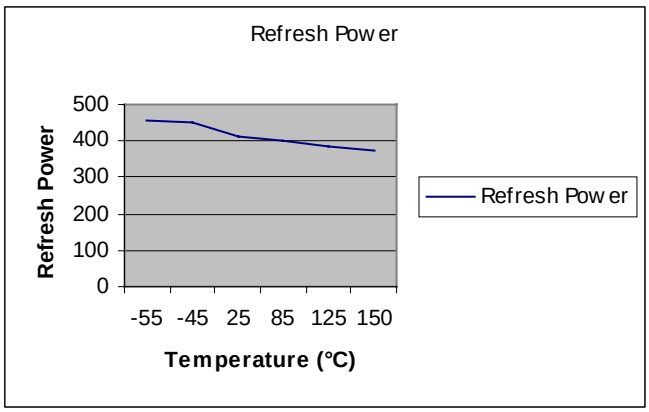


Figure 4-20 : Mode Auto-refresh.

4.5.4.2 Simulation

Les tableaux suivants et leurs courbes associées présentent les résultats en fonction de la température :

- Des courants maximum simulés
- Des puissances moyennes calculés (= intégration des courants instantanés)
- Des coefficients de variation avec la température
- Concernant 256 cellules mémoires

temperature	average power	max current	variation coeff.
-50	380	418	
-25	384	403	-3,59%
0	393	387	-3,97%
25	406	378	-2,33%
50	422	369	-2,38%
75	440	357	-3,25%
100	461	352	-1,40%
125	481	346	-1,70%
150	504	339	-2,02%
Average variation coefficient			-2,58%

Tableau 4-4 : Résultats de la simulation de la cellule mémoire SDRAM.

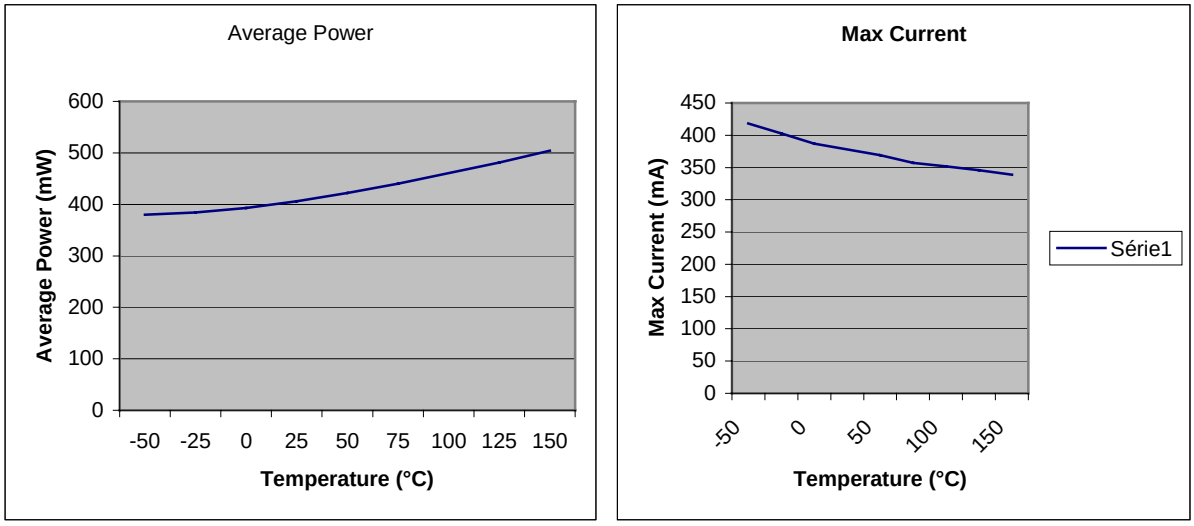


Figure 4-21 : Résultats de la simulation de la SDRAM.

4.5.5 Discussion des résultats

4.5.5.1 Courants mesurés et puissance calculée

a) Les mesures de courants ont été traitées
- pour une gamme de température [-50°C → 150°C]
- dans trois modes:
  - Operating mode
  - Stand-by mode
  - Auto Refresh mode tous les 4096 cycles
- à une fréquence plus basse que celle normalement utilisée
  - Cela donne des valeurs de puissance plus basse que celles attendues

b) La puissance peut être calculée en utilisant l'expression suivante:

Puissance = Courant mesuré * VCC

c) En comparant les valeurs de courant, ou de puissance dans les trois modes

- Le mode Auto Refresh fournit la plus grande valeur, mais comme il est effectué tous les 4096 cycles, le poids de ce mode dans une séquence comprenant les trois modes n'apporte que du bruit
- Comme les valeurs du mode Stand-by sont les plus basses, le mode Opérationnel est le plus significatif et de ce fait peut être comparé avec les valeurs simulées

d) Les valeurs de courant et de puissance sont décroissantes quand la température croît de -50°C à +150°C avec un coefficient moyen de variation de –2,65%

4.5.5.2 Simulation

a) Les simulations ont été effectuées sur la même gamme de température [-50°C → 150°C] :

- Adressant 256 cellules mémoires (256 bits), c'est ce que nous avons déterminé,

- Avec une séquence de test décrite dans la figure 4-15,
- Où les courants max apparaissent dans la figure 4-16 au sommet des pics de courant, et détectés par VamSpiceDesigner en tant que MaxCurrent
- Où la puissance moyenne est calculée
    - En intégrant les puissances instantanées (Figure 4-17)
    - En prenant les coordonnées du point final apparaissant dans la figure 4-17 : MaxEnergy et MaxTime (durée de la séquence) :
        - Puissance moyenne = MaxEnergy / MaxTime

b) En comparant la puissance moyenne simulée et l'évolution des courants max on constate que :

- La puissance moyenne croit, mais le courant max décroît

b) En comparant les courants max mesurés et simulés on constate que :

- Ils évoluent dans le même sens (décroissant) avec un coefficient moyen de variation (-2,58% dans le cas de la simulation). La simulation faite quand on considère l'amplification après 10 lectures donne également pratiquement le même coefficient de variation (= -2,65%).

4.5.6 Utilisation du calculateur de puissance de Micron™

Une autre façon de déterminer la puissance moyenne est d'utiliser le calculateur de puissance MICRON. Les conditions d'utilisation doivent être définies dans la figure 4-22.

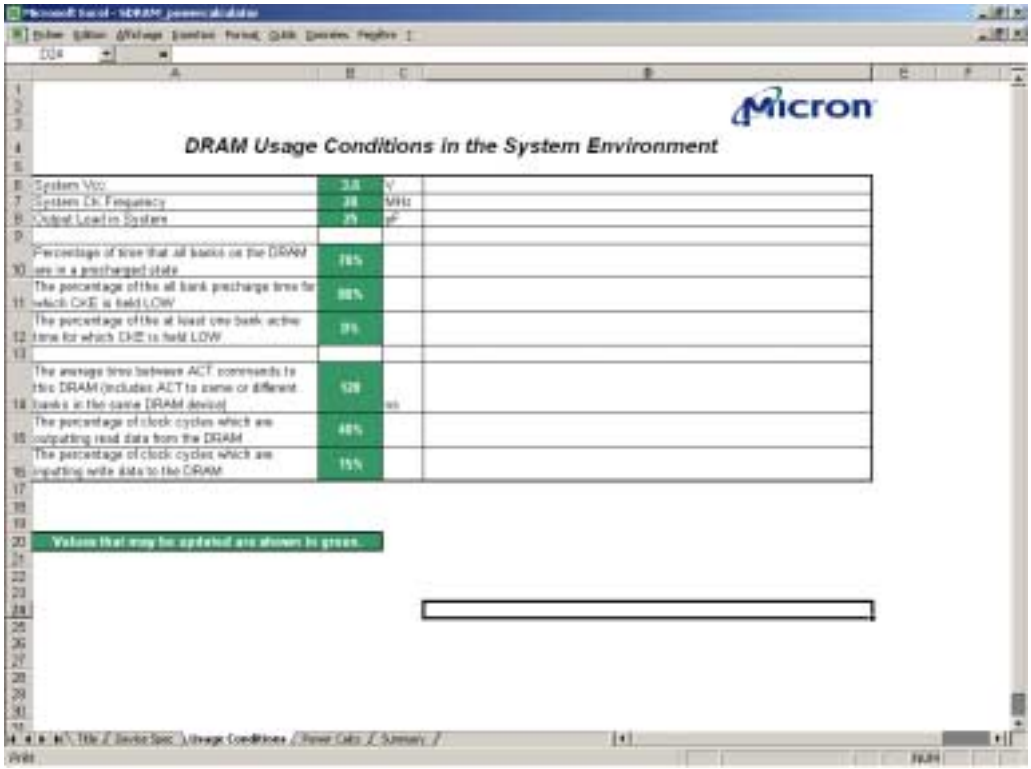


Figure 4-22 : Conditions d'utilisation du calculateur de puissance MICRON.

Il est difficile de définir les conditions d'utilisation, mais cela peut donner quelque estimation de cette puissance moyenne.

La figure 4-23 présente une synthèse des résultats de calcul.

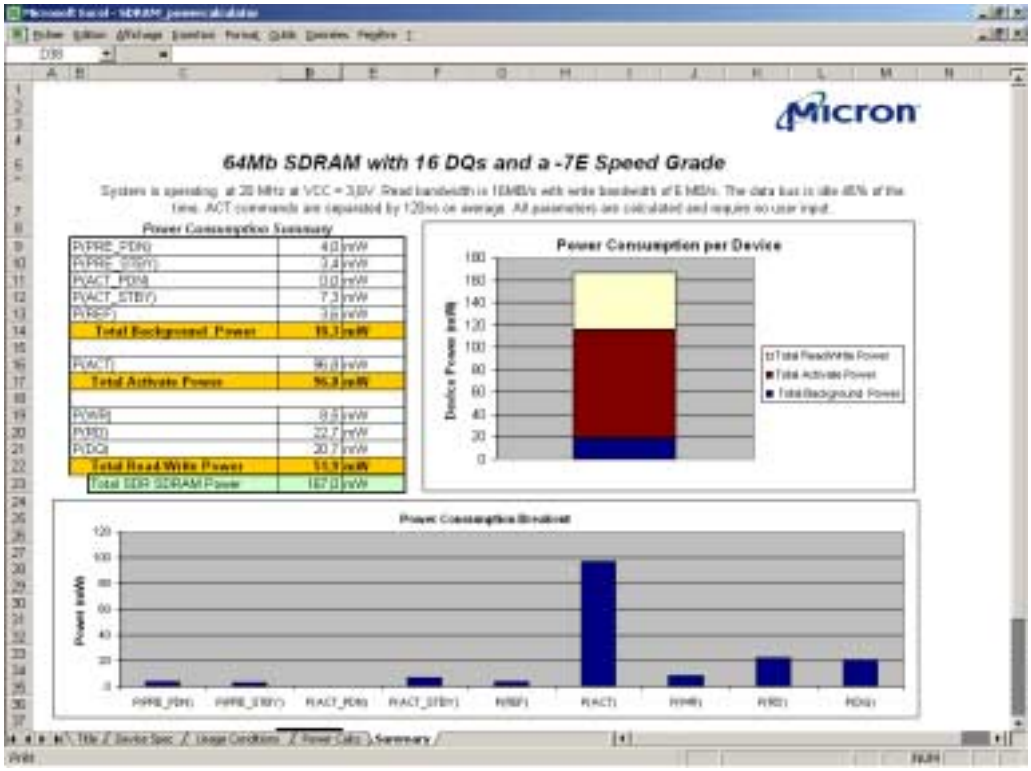


Figure 4-23 : Résultats de calcul du calculateur MICRON SDRAM Power.

Cependant, il n'est pas possible d'estimer avec ce calculateur l'évolution de la puissance moyenne avec la température. En appliquant le coefficient de variation issu de la simulation, on peut alors estimer la dépendance de la puissance moyenne avec la température.

Variation factor	-0,0265	determined from measurements
power at 25°C mW)	167,00	calculated at 25°C

temperature (°C)	power (mW)
-50	154,07
-25	158,27
0	162,57
25	167,00
50	171,43
75	175,97
100	180,63
125	185,42
150	190,33

Tableau 4-5 : Puissance moyenne calculée en fonction de la température.

4.5.7 Conclusions et Commentaires sur la mesure et la simulation

a) Les mesures ont été effectuées à une fréquence beaucoup plus basse que la fréquence maximum autorisée pour ce type de mémoire (133MHz) à cause de la limitation en fréquence de l'appareil de mesure (20 MHz).

b) La conclusion intéressante à noter est que nous devons comparer l'évolution des courants max mesurés et simulés en fonction de la température qui est pratiquement la même. L'appareil de mesure ne mesure que les courants, de telle sorte que les puissances doivent être calculées en multipliant ces courants par la tension VCC appliquée.

c) Cette évolution avec la température est cependant très faible, conséquence de la conception de la mémoire SDRAM.

d) Il est impossible de déterminer par simulation la puissance effectivement consommée par le circuit SDRAM, parce que nous ne savons pas le nombre d'éléments mémoires adressés dans une séquence mesurée donnée.

e) De plus, les simulations ne prennent pas en compte les commandes de contrôle fournis par les circuits logiques périphériques, car nous n'avons pas d'information à leur sujet, et nous ne pouvons pas évaluer leur importance relative.

f) En combinant la simulation pour déterminer le coefficient de variation de la puissance moyenne avec la température, et le calcul de la puissance avec le calculateur MICRON, il est possible d'obtenir la puissance moyenne en fonction de la température.

4.6 Conclusion

Ce chapitre consacré au traitement électro-thermique des mémoires FLASH et SDRAM a montré l'intérêt que peut présenter la conception schématique hiérarchique associée à la modélisation et la simulation comportementale dans un domaine où une modélisation classique est insuffisante.

L'étude de l'auto-échauffement, tenté avec ADVanceMS de Mentor Graphics en utilisant un réseau thermique $R_{th}C_{th}$, semble inadaptée compte tenu de la différence des constantes de temps électriques et thermiques, ce qui amène à l'étude en deux étapes : l'une « électrique » qui conduit au calcul des puissances et des énergies des circuits électriques sans tenir compte de l'auto-échauffement, à l'aide de la simulation électrique, l'autre « thermique » qui part de ces résultats pour calculer les températures, à l'aide de la simulation thermique. Force est de constater cependant que la simulation thermique est rendue difficile du fait des faibles consommations, et de leur très faible variation en fonction de la température ambiante, même dans la gamme étendue [–50°C.. +125°C], de ces types de composants que sont les mémoires étudiées.

Pour l'études de cellules mémoires SDRAM et FLASH, nous n'avons pas poussé l'analyse et le test jusqu'au bout car nous n'avons pas accès à la technologie de circuit de commande et de contrôle (décodeur, logique de commande) et de plus nous n'avons pas les modèles SPICE en provenance du fondeur.

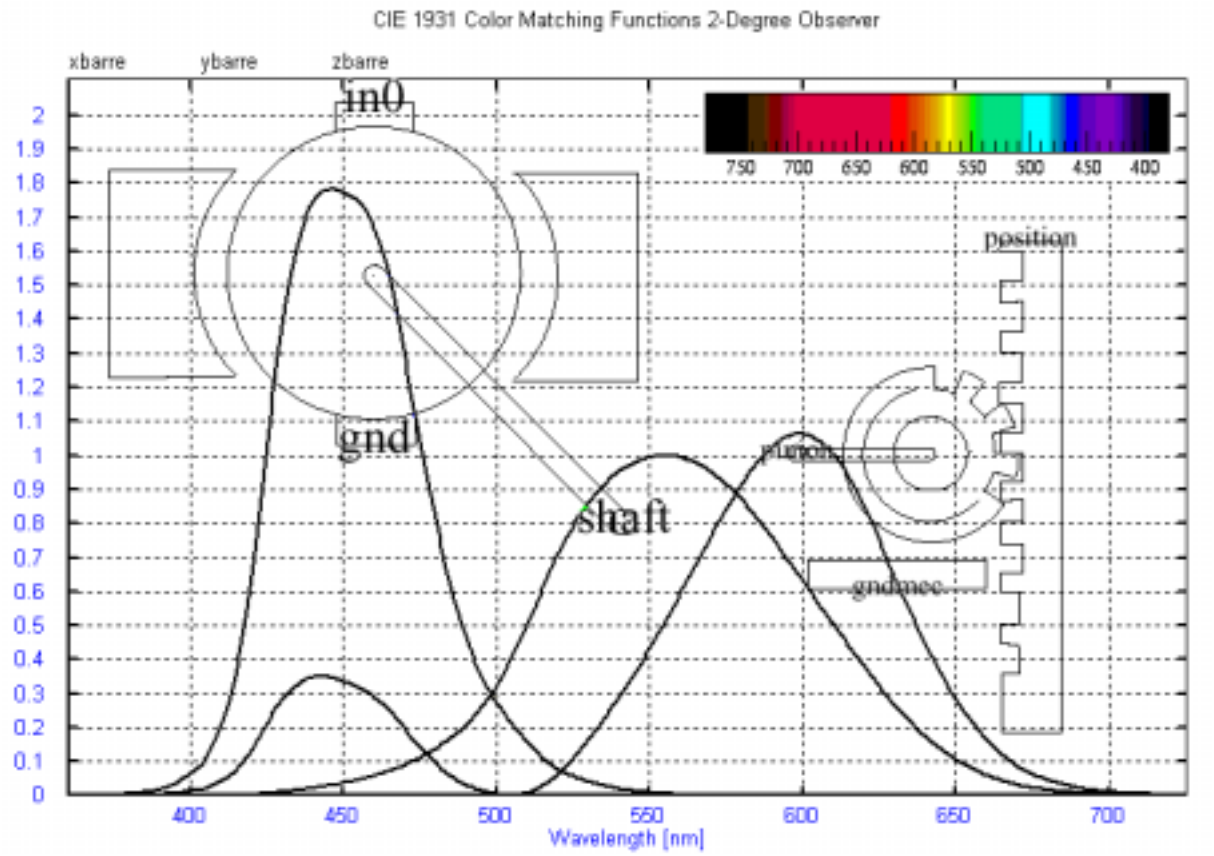
Dans le cas de la SDRAM, la faible dissipation de la puissance, due d'une part aux limitations de la fréquence de mesure (20MHz au lieu de 133MHz), et d'autre part au fait que le modèle

électrique ne tient pas compte des circuits périphériques de commande pour évaluer réellement l'effet de la température, a pour conséquence de ne pas permettre d'apprécier l'effet de la température sur la performance.

Pour le cas de la cellule mémoire FLASH, les méthodes proposées dans le projet SPARTE, ne sont pas utilisables pour ce type de composant car la dissipation en terme de puissance est très faible (~50mW). Dans ces conditions l'étude thermique déduite de l'effet électrique est largement suffisante.

5

MODELISATION - SIMULATION DE SYSTEMES MULTI-TECHNOLOGIQUES



Chapitre 5

5 MODELISATION – SIMULATION DE SYSTEMES MULTI-TECHNOLOGIQUES

5.1 Introduction

Dans la première partie et dans le chapitre 2, nous avons illustré quelques exemples test de notre outil VamSpiceDesigner qui sont des exemples analogiques (et mixtes) dans des niveaux d'abstraction différente. Nous allons montrer maintenant dans ce chapitre qu'il est possible d'utiliser cet outil comme un outil multi-technologique, c'est-à-dire dans des domaines technologiques très différents, en présentant deux applications :

- Un modèle colorimétrique,
- Un modèle électro-mécanique.

Nous allons montrer dans la première partie de ce chapitre les bases essentielles de la colorimétrie nécessaires à la compréhension de notre exemple. Nous prenons comme cas d'étude la détermination de la couleur rendue d'un objet qui a une certaine couleur intrinsèque à travers le brouillard et à une certaine distance.

Dans la deuxième partie, nous analysons un exemple d'un système électro-mécanique en traitant le cas d'un élévateur de siège de voiture.

5.2 Modèles colorimétriques

5.2.1 VHDL-AMS pour la Colorimétrie

5.2.1.1 Introduction

La colorimétrie est la science de mesure des couleurs. Chacun de nous peut observer la distribution spectrale des couleurs différemment, c'est pourquoi la CIE (Commission Internationale de L'Eclairage)[WWW3] a défini des standards d'observation. Ces ensembles de standards sont des données numériques expérimentales qui sont appelées couleurs de mesure (***color matching***). Elles représentent trois sources de couleur particulières qui émettent de la lumière sur un écran blanc. Ces trois projections se chevauchent et forment un mélange de couleurs. L'intensité des trois sources de couleur doit être modifiée une à une de façon à avoir tous les mélanges possibles. Les trois sources

de couleur qui permettent d'effectuer cette expérience sont le rouge, le vert, et le bleu dont les longueurs d'onde λ respectives sont 700nm, 546.1nm, et 435.8nm [WYS82]. Les courbes de couleurs de mesure proposées par CIE sont montrées sur la figure 5-2. Ces fonctions normalisées sont notées $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, et $\bar{z}(\lambda)$.

La CIE a défini deux standards d'observation qui sont connus sous le nom **CIE 1931** et **CIE 1964** (Figures 5-2 et 5-3). Le premier est un standard d'observation à 2 degrés qui est basé sur les travaux de Guild [WWW2][GUI31] et Wright [WRI28] (Figure 5-1). Le deuxième est un standard d'observation à 10 degrés qui est basé sur les travaux de Stiles et Burch [STI55], et Speranskaya [WWW2] (Figure 5-1). Le standard CIE 1931 est utilisé dans des milieux lumineux (lumière de jour par exemple), par contre le standard CIE 1964 est utilisé dans des milieux sombre (la nuit par exemple).

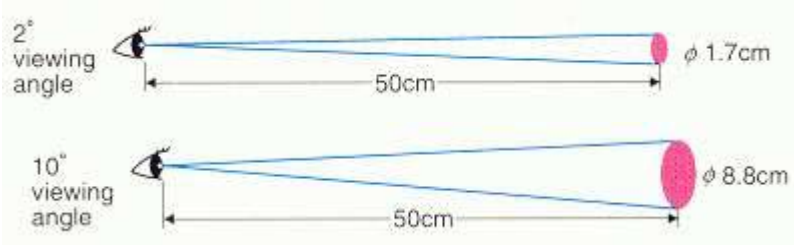


Figure 5-1 : Les angles d'observation à 2 et 10 degrés.

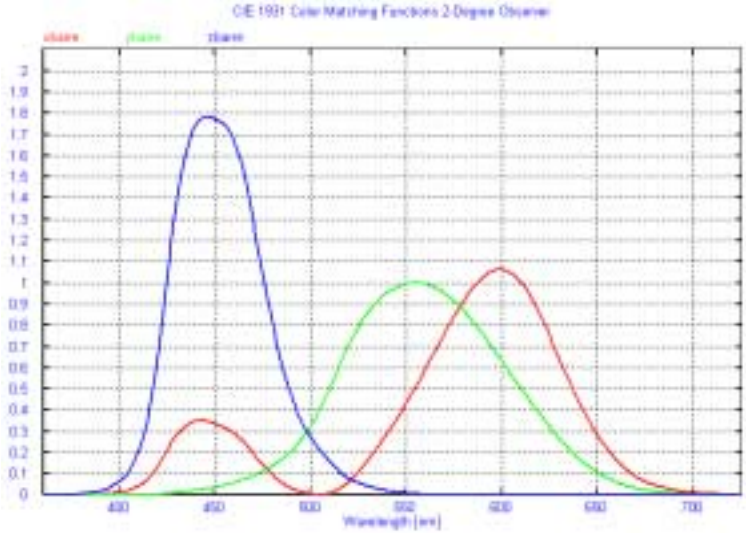


Figure 5-2 : Courbes ($\bar{x}(\lambda)$, $\bar{y}(\lambda)$, et $\bar{z}(\lambda)$) de *color matching* de CIE 1931 à 2 degrés.

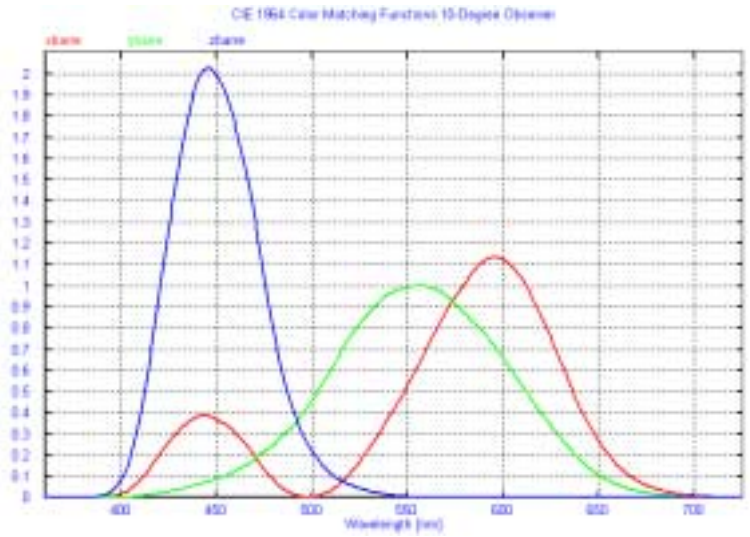


Figure 5-3 : Courbes ($\bar{x}(\lambda)$, $\bar{y}(\lambda)$, et $\bar{z}(\lambda)$) de *color matching* de CIE 1964 à 10 degrés.

5.2.1.2 Les bases de la colorimétrie

5.2.1.2.1 Définition des tristimuli

Les tristimuli X, Y et Z, qui représentent les énergies reçues par l'œil, sont calculés à partir de ces équations [CHA98][CHA99] :

$$X = \int \bar{x}(\lambda).S(\lambda).R(\lambda).d\lambda \quad (5.1)$$

$$Y = \int \bar{y}(\lambda).S(\lambda).R(\lambda).d\lambda \quad (5.2)$$

$$Z = \int \bar{z}(\lambda).S(\lambda).R(\lambda).d\lambda \quad (5.3)$$

Avec :

$S(\lambda)$, la luminance relative de la source

$R(\lambda)$, la distribution spectrale de la réflectance intrinsèque ou aussi la transmittance.

Les variables X, Y, et Z définissent les luminosités des couleurs dont l'ensemble est connu sous le nom **CIE XYZ**. Le CIE XYZ est un espace linaire à trois dimensions. Si nous projetons cet espace sur un plan, nous aurons $X+Y+Z=1$. Le résultat dans un espace à deux dimensions est connu sous le nom de diagramme chromatique. Les coordonnées dans cet espace sont souvent appelées **x** et **y** qui peuvent être exprimées à partir des expressions suivantes :

$$x = \frac{X}{X+Y+Z} \quad (5.4)$$

$$y = \frac{Y}{X+Y+Z} \quad (5.5)$$

$$z = \frac{Z}{X+Y+Z} = 1-x-y \quad (5.6)$$

5.2.1.2.2 Caractéristique de l'écran d'affichage et la conversion XYZ↔ RGB

La caractéristique de l'écran d'affichage est déterminée par trois couleurs primaires : le rouge, le vert, et le bleu, notés par **R** (*red*), **G** (*green*), et **B** (*blue*). En pratique, les valeurs de RGB se situent entre 0 et 255 (codage sur 8 bits) pour une seule couleur, d'où en totalité, 3 x 256 couleurs possibles (codage sur 24 bits).

Les couleurs de base sont spécifiées par les coordonnées des tristimuli. Par exemple, la couleur rouge (R) est spécifiée par X_R, Y_R, et Z_R. Cependant, les coordonnées des tristimuli X, Y, et Z sont une combinaison linaire des couleurs RGB, et données par la matrice linaire suivante :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{pmatrix} . \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{5.7}$$

La couleur **blanc** est une autre caractéristique d'un écran d'affichage. Elle est normalisée par :

$$R_W = G_W = B_W = 1 \tag{5.8}$$

De même on a :

$$X_W = X_R + X_G + X_B \tag{5.9}$$

Pour chercher les coordonnées de RGB à partir des tristimuli XYZ, nous pouvons inverser la matrice de l'équation 5.10.

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} R_X & R_Y & R_Z \\ G_X & G_Y & G_Z \\ B_X & B_Y & B_Z \end{pmatrix} . \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{pmatrix}^{-1} . \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{5.10}$$

En pratique, au lieu d'utiliser les coordonnées des tristimili, nous utilisons les coordonnées chromatiques (x_R, y_R), (x_G, y_G), et (x_B, y_B), et de même pour les coordonnées du blanc (x_W, y_W). Pour chercher les transformations chromatiques, les équations décrites en 5.4, 5.5, et 5.6 resteront valables.

Donc, chaque système d'affichage est caractérisé par ses coordonnées chromatiques. Le plus connu de ces systèmes est le CIE D_{65} [WWW3] qui est un standard qui caractérise la lumière du jour et qui a les coordonnées chromatiques suivantes :

$$x_W = 0.3127, y_W = 0.3290 \tag{5.11}$$

$$\begin{pmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \end{pmatrix} = \begin{pmatrix} 0.64 & 0.30 & 0.15 \\ 0.33 & 0.60 & 0.06 \end{pmatrix} \tag{5.12}$$

Nous pouvons citer d'autres systèmes d'affichage utilisés dans la technologie de fabrication des écrans de téléviseurs ou d'ordinateurs:

- **PAL/SECAM** : **Phase Alternating Line / SEquential Couleur Avec Mémoire** est un standard européen (Franco-allemand).
- **NTSC** : ***National Television System Committee*** est un standard américain.
- **EBU** : ***European Broadcasting Union*** est un standard européen qui utilise l'illuminant CIE D_{65}.

- **SMPTE** : ***The Society of Motion Picture and Television Engineers*** qui utilise l'illuminant CIE D_{65}.

5.2.1.2.3 La correction GAMMA

La correction Gamma consiste à utiliser un paramètre numérique qui représente la non-linéarité de la production de l'intensité lumineuse. Son utilité s'explique par la présence de divers phénomènes ne relevant pas seulement de la physique mais aussi de la perception visuelle humaine.

La correction apportée par le facteur Gamma est la transformation d'un système linaire de coordonné R, G, et B en un système non-linaire de coordonnées R', G', et B'. Cette transformation expérimentale est de la forme : [WWW4]

$$R'=\begin{cases} R & (|Gamma|-1<0.0001 \\ R^{\frac{1}{Gamma}} & (|Gamma|-1\geq 0.0001 \end{cases} \tag{5.13}$$

Et de même pour G' et B'.

L'instance internationale de normalisation recommande de prendre, pour cette correction une fonction puissance d'exposant 0.45. Car ce facteur est un bon compromis à la fois pour les tubes cathodiques et pour la perception visuelle. Nous aurons, donc, comme valeur de Gamma = 2.2 :

$$Gamma=\frac{1}{0.45}\approx 2.2 \tag{5.14}$$

5.2.1.3 Modélisation VHDL-AMS

Le cas d'étude que nous nous proposons de décrire est celui de la détermination de la couleur rendue d'un objet d'une certaine couleur intrinsèque à travers le brouillard. Ce qui se traduit par : **un objet de couleur spectral C1 est illuminé par un illuminant spécifique (S) et est observé à travers le brouillard de densité k et à une distance D** (Figure 5-4).

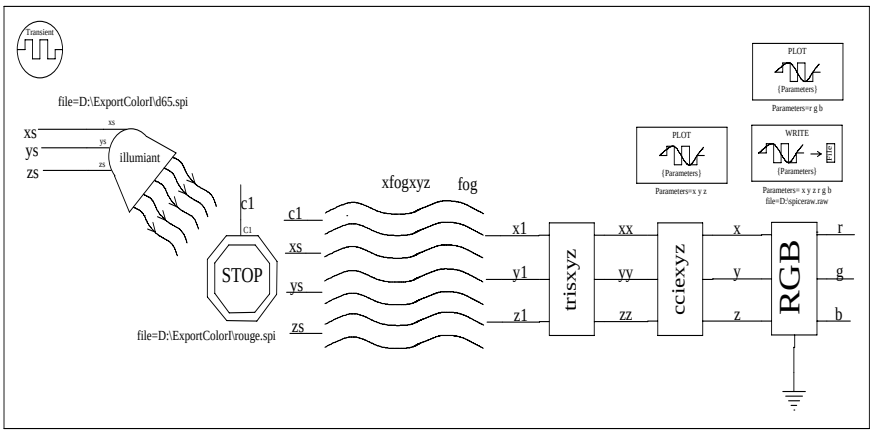


Figure 5-4 : Le système colorimétrique à étudier.

Dans cet exemple, nous utilisons la modélisation (ou la spécification) schématique pour modéliser les différentes parties du système colorimétrique.

Modèle de l'effet du brouillard (*Fog effects*)

D'après les lois de *Beer* développés dans l'article [WILL87], le modèle du brouillard est donné par l'expression suivante : [CHA98][CHA99]

$$R(\lambda)_{fog} = S(\lambda).\left(1-\left(1-R(\lambda)\right)e^{-k.D}\right) \tag{5.15}$$

Où :

k (m^{-1}) est la densité de brouillard,
D (m) est la distance entre l'observateur et l'objet,
R(λ) est la réflectance de l'objet,
S(λ) est l'intensité de la source lumineuse,
$R(\lambda)_{fog}$ est la réflectance totale du panneau plus l'effet du brouillard.

Nous commençons par définir l'icône de notre modèle du brouillard (Figure 5-5). Ensuite, nous générons automatiquement le modèle VHDL-AMS (voir chapitre 2) et nous ajoutons le modèle du brouillard décrit dans l'équation 5.15 (Figure 5-6).

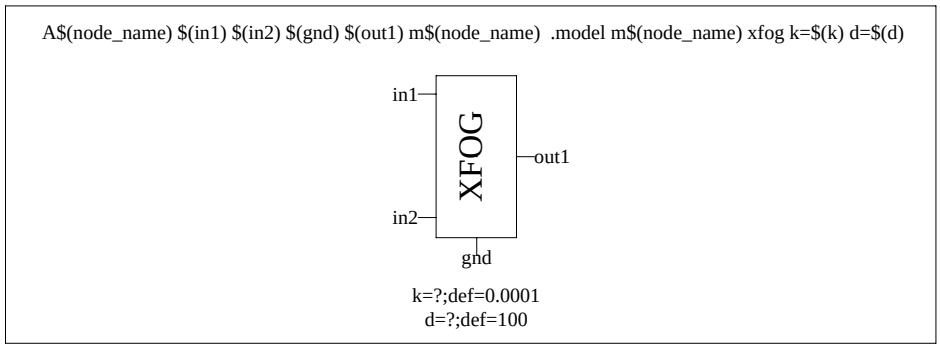


Figure 5-5 : Icône du modèle de brouillard.

```
-- VHDL-AMS automatically generated from facet xfog{ic}
ENTITY xfog IS
 GENERIC (d :real:= 100.0; k :real:= 0.0001);
 PORT (TERMINAL in1, in2, gnd: Electrical; QUANTITY out1: OUT Real);
END xfog;
ARCHITECTURE xfog_BODY OF xfog IS
 quantity vinxyz across in1 to gnd;
 quantity vinr across in2 to gnd;
BEGIN
  out1 == vinxyz * (1.0 - (1.0 - vinr) * exp(-k*d));
END xfog_BODY;
```

Figure 5-6 : Modèle VHDL-AMS du brouillard.

Le modèle VHDL-AMS décrit dans la figure 5-6 est un modèle unidimensionnel. Pour former les trois voies de tristimuli, nous utilisons le modèle structurel de la figure 5-7. A partir de ce modèle structurel, nous générons l'icône du modèle global.

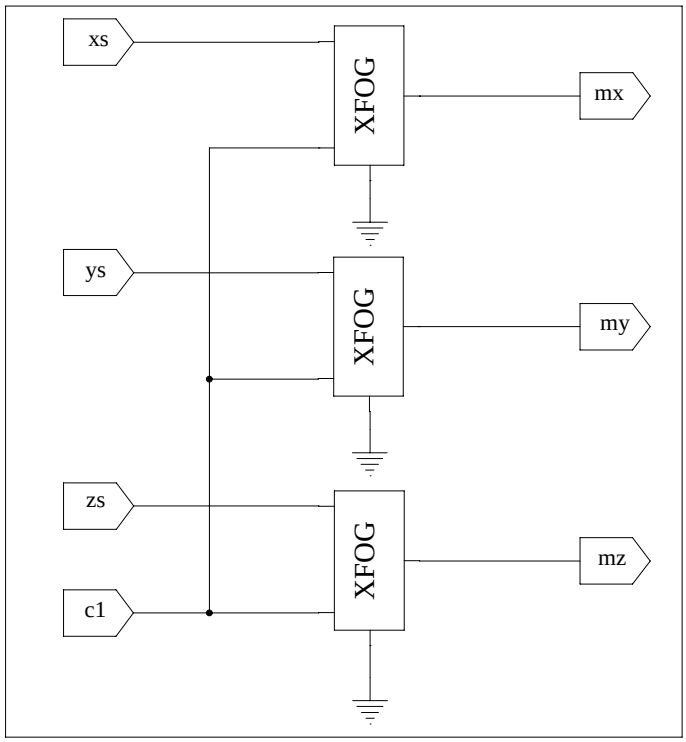


Figure 5-7 : Modèle structurel du brouillard.

Modèle des tristimuli et le modèle des coordonnées chromatiques

Les tristimuli sont modélisés à partir des équation 5.1, 5.2, ou 5.3. Pour calculer l'intégrale, nous pouvons utiliser trois inductances ou trois capacités en série [CHA99] [OAL98] qui représentent les tristimuli X, Y, et Z (Figure 5-8).

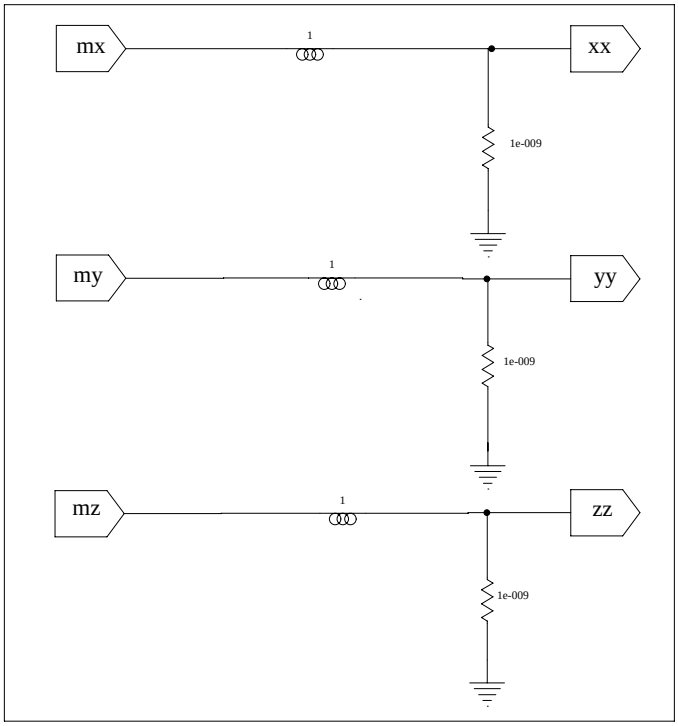


Figure 5-8 : L'intégration du courant pour calculer les tristimuli X, Y, et Z.

Le modèle des coordonnées chromatiques élaboré à partir de l'équation 5.1 est donné par :

```
-- VHDL-AMS automatically generated from facet ccie{ic}
ENTITY ccie IS
  -- GENERIC (--Empty generic
  --               );
 PORT (TERMINAL in1, in2, in3, gnd: Electrical;
          QUANTITY out1: OUT Real);
END ccie;
ARCHITECTURE ccie_BODY OF ccie IS
 quantity vin1 across in1 to gnd;
 quantity vin2 across in2 to gnd;
 quantity vin3 across in3 to gnd;
BEGIN
  out1 == vin1 / (vin1 + vin2 + vin3 + 1.0e-10);
END ccie_BODY;
```

Modèle de la conversion XYZ vers RGB

Les équations de ce modèle de conversion se déduisent des équations 5.7 à 5.10. Ceci est présenté par le code VHDL-AMS suivant :

```
-- VHDL-AMS automatically generated from facet crgb{ic}
ENTITY xyztorgb IS
  GENERIC(
                xred      :real:=0.0;yred     :real:=0.0;
                xgreen    :real:=0.0;ygreen   :real:=0.0;
                xblue     :real:=0.0;yblue    :real:=0.0;
                gamma     :real:=2.2;
                minv      :real:= 0.0;maxv    :real:= 1.0
              );
  PORT(TERMINAL nx, ny, nz, ref : electrical);
  PORT(QUANTITY rr : OUT real);
  PORT(QUANTITY gg : OUT real);
  PORT(QUANTITY bb : OUT real);
END ENTITY xyztorgb;

ARCHITECTURE xyztorgb_BODY OF xyztorgb IS
--*****************************************************************
FUNCTION max(aa, bb : real) RETURN real IS
VARIABLE value : real;
BEGIN
 IF (aa > bb) THEN
  value := aa;
 ELSE
  value := bb;
 END IF;
 RETURN value;
END FUNCTION;
--*****************************************************************
FUNCTION limitrgb(value, low, high :real) RETURN real IS
VARIABLE value_out : real;
BEGIN
 IF (value < low) THEN
    value_out := low;
     ELSE IF (value > high) THEN
            value_out := high;
     ELSE
            value_out := value;
```

```
    END IF;
 END IF;
 RETURN value_out;
END FUNCTION;
--*************************************************************
 QUANTITY xr, yr, zr :real;
 QUANTITY xg, yg, zg :real;
 QUANTITY xb, yb, zb :real;

 QUANTITY d, r1, g1, b1 :real;
 QUANTITY max1, maxRGB: real;

 QUANTITY       xc across nx to ref;
 QUANTITY       yc across ny to ref;
 QUANTITY       zc across nz to ref;

BEGIN
  xr == xred;
  yr == yred;
  zr == 1.0 - xr - yr;

  xg == xgreen;
  yg == ygreen;
  zg == 1.0 - xg - yg;

  xb == xblue;
  yb == yblue;
  zb == 1.0 - xb - yb;

 d  == xr*yg*zb - xg*yr*zb - xr*yb*zg + xb*yr*zg + xg*yb*zr - xb*yg*zr;
 r1 ==  (-xg*yc*zb + xc*yg*zb + xg*yb*zc - xb*yg*zc - xc*yb*zg + xb*yc*zg) / d;
 g1 ==  ( xr*yc*zb - xc*yr*zb - xr*yb*zc + xb*yr*zc + xc*yb*zr - xb*yc*zr) / d;
 b1 ==  ( xr*yg*zc - xg*yr*zc - xr*yc*zg + xc*yr*zg + xg*yc*zr - xc*yg*zr) / d;

 r1 == limitrgb(r1, minv, maxv);
 g1 == limitrgb(g1, minv, maxv);
 b1 == limitrgb(b1, minv, maxv);

 max1  == max(r1, g1);
 maxRGB  == max(max1, b1);

-- Gamma correction
IF ((abs(gamma1) - 1.00) < 0.0001) USE
   rr    == 255 * r1 / maxRGB;
   gg  == 255 * g1 / maxRGB;
   bb  == 255 * b1 / maxRGB;
 ELSE
   rr  == (255 * r1/maxRGB)**(1/gamma1);
   gg == (255 * g1/maxRGB)**(1/gamma1);
   bb == (255 * b1/maxRGB)**(1/gamma1);
 END USE;
END xyztorgb_BODY;
```

5.2.1.4 Simulation du système colorimétrique

Dans notre cas d'étude, nous prenons un panneau de couleur rouge et un illuminant de type CIE D_{65} (*daylight*). La distribution spectrale du panneau se trouve dans le fichier SPICE *rouge.spi*.

Le fichier d65.spi, contient le spectre de couleurs de mesure (CMF : *color matching function*) $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, et $\bar{z}(\lambda)$.

Le fichier SPICE de simulation est généré automatiquement à partir du schéma d'ensemble du système (Figure 5-4). Nous générons un fichier de type *SPICERAW* qui contient les données ou les résultats de la simulation. Nous récupérons ensuite, ces données et nous les traitons avec notre programme **Colorimetric v1.03**[3] [JECH03].

La simulation est effectuée à une distance D = 10m et avec une densité spectrale du brouillard de valeur k = 0.0004 m^{-1}. De même, nous prenons comme système d'affichage le SMPTE[4] et Gamma de valeur 2.2.

La figure 5-9 présente le résultat de simulation du système avant d'attaquer le modèle de conversion de x, y, et z vers RGB. Nous pouvons effectuer la conversion directement avec Colorimetric en cochant ***XYZ to RGB*** (Figure 5-10) avant d'afficher les résultats. La deuxième possibilité, est d'utiliser directement le modèle décrit en VHDL-AMS (r, g, et b) et d'afficher directement les résultats, c'est-à-dire, les courbes et couleurs (Figure 5-11). Dans la figure 5-11, nous essayons de ramener toutes les courbes au maximum des valeurs de couleur c'est-à-dire à 255. Les couleurs rendues dans la figure 5-11 sont à une dimension ; nous pouvons visualiser les mêmes couleurs en affichant la courbe à deux dimensions, et nous obtenons la figure 5-12. La couleur rendue est celle obtenue à la fin de l'intégration, car les couleurs affichées auparavant sont les couleurs intermédiaires en cours d'intégration. La figure 5-12 nous renseigne donc sur les couleurs rendues par rapport à l'enveloppe standard des couleurs (« sabot de cheval »). Nous répétons la même opération pour plusieurs distances et nous enregistrons les couleurs rendues pour les envoyer vers un panneau de signalisation (Figure 5-13).

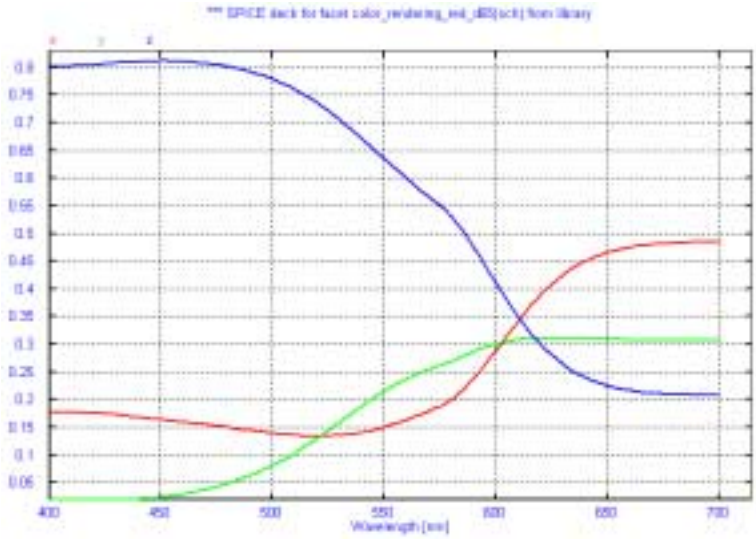


Figure 5-9 : Coordonnées chromatiques x, y, et z.

[3] **Colorimetric v1.03** est un outil conçu et développé à ENST Paris. Son rôle est d'afficher à partir des fichiers SPICE (*SPICERAW*) les couleurs rendues par un système colorimétrique. A la fin du traitement, ces couleurs sont envoyées vers des panneaux de signalisation pour comparer les nuances des couleurs.
[4] SMPTE : *The Society of Motion Picture and Television Engineers*.

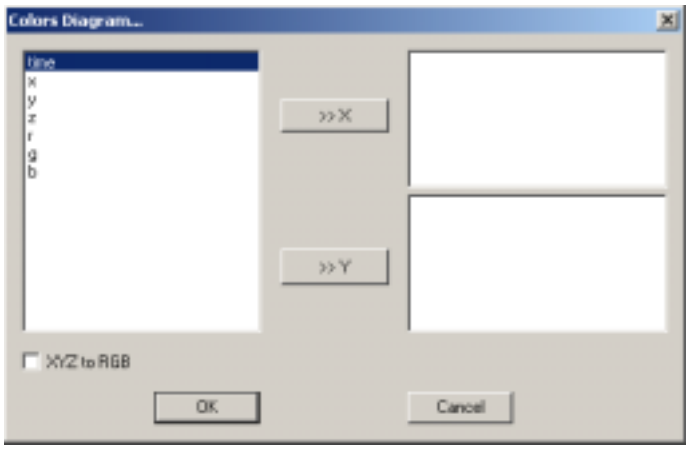


Figure 5-10 : Extrait d'une fenêtre de l'outil Colorimetric v1.03. Sélection des paramètres à afficher et la conversion *XYZ to RGB*.

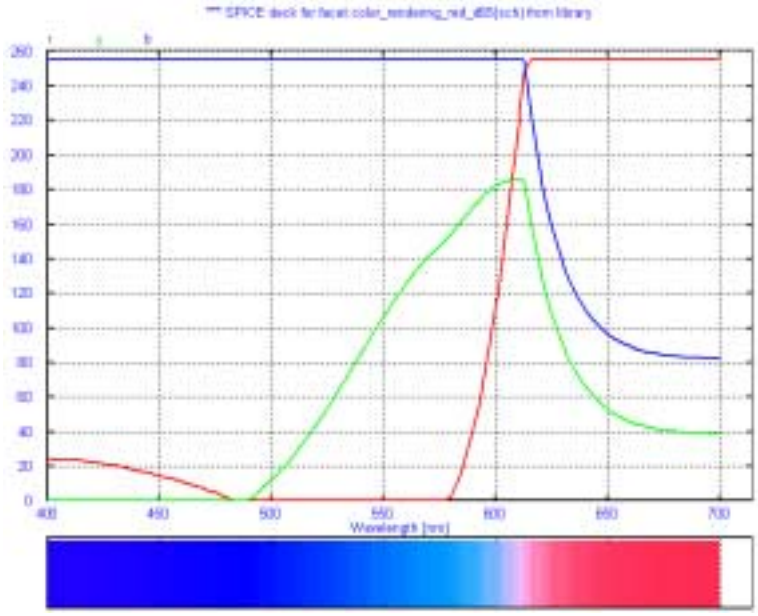


Figure 5-11 : Transformation des coordonnées chromatiques x, y, et z vers RGB et les couleurs correspondantes.

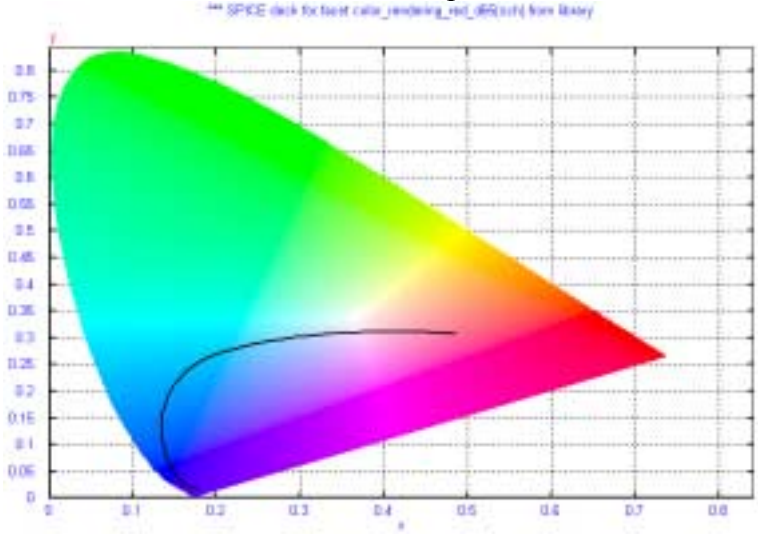


Figure 5-12 : La position de couleur rendue dans l'enveloppe standard (*horseshoe*) [JEM03].

Figure 5-13 : Couleurs rendues pour plusieurs distances (à partir du bas) [1000, 700, 400, 10 m] avec une densité de brouillard de 0.0004m^{-1}.

5.3 Modélisation-simulation électro-mécanique

Cette application est inspirée de l'exemple décrit dans [MAN95]. Elle présente l'analyse du comportement d'un système électromécanique d'élévation du siège d'une voiture commandé à l'aide d'un bouton pressoir. Le schéma bloc de la figure 5-14 contient les différents éléments du système modélisés chacun avec VHDL-AMS : un moteur à courant continu (*DC motor*), un système d'engrenages, un convertisseur de vitesse et un ensemble pignon-crémaillère qui supporte le siège.

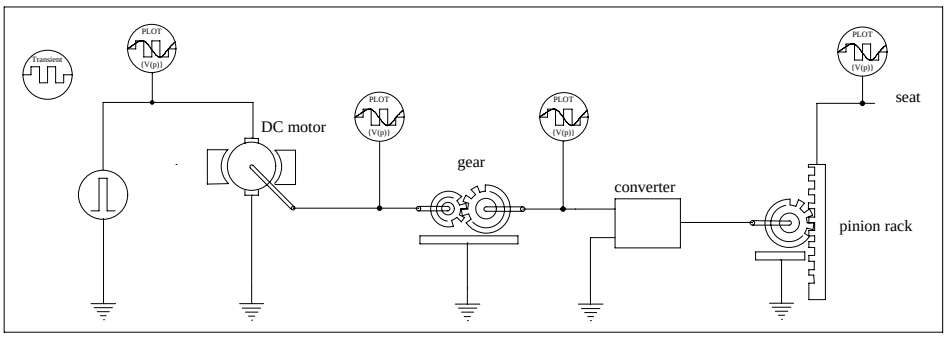


Figure 5-14 : Schéma bloc de l'élévateur de siège.

Les icônes et les modèles VHDL-AMS associés à ces différents éléments sont décrits dans l'annexe 5. Cependant, à titre d'exemple et pour rappeler la méthode déjà utilisée dans la section précédente, les figures 5-15 et 5-16 montrent les icônes du moteur à courant continu et de l'ensemble pignon-crémaillère.

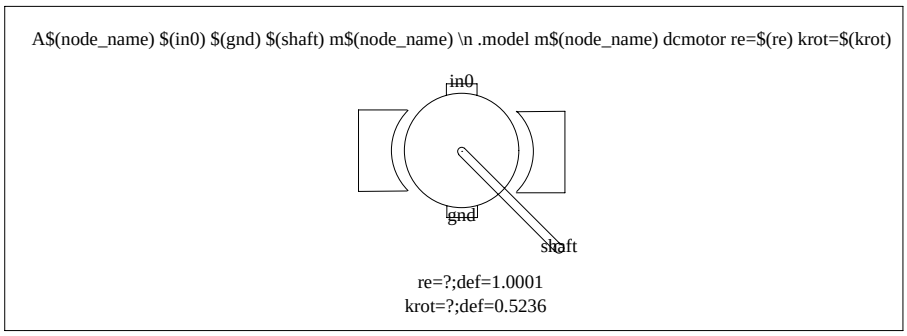


Figure 5-15 : Icône du moteur à courant continu.

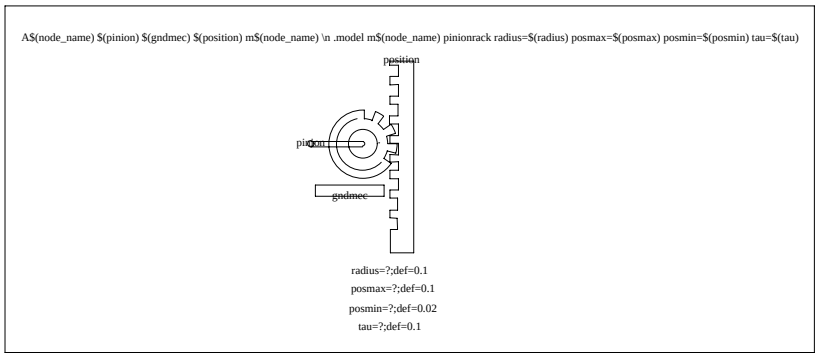


Figure 5-16 : Icône de l'ensemble pignon crémaillère.

A partir de l'icône pignon-crémaillère le squelette du modèle VHDL-AMS est généré automatiquement et complété comme indiqué sur la figure 5-17.

```
-- VHDL-AMS automatically generated from facet pinionrack{ic}
ENTITY pinionrack IS
 GENERIC (tau :real:= 0.1; posmin :real:= 0.02; posmax :real:= 0.1;
 radius :real:= 0.1);
 PORT (TERMINAL pinion, gndmec: Mecanical; QUANTITY position: OUT
 Real);
END pinionrack;
ARCHITECTURE pinionrack_BODY OF pinionrack IS
 quantity apos, pos1 : real;
 quantity piangle across pos through pinion to gndmec;
BEGIN
 apos == radius * piangle;
 pos1 == 0.9*posmax;
 if (apos < posmin) USE
    pos == posmin;
 else if (apos > pos1) use
         pos == 0.9 * posmax + 0.1 * posmax * (1.0 - exp((0.9 * posmax - apos) / tau));
       else  pos == apos;
       end use;
 end use;
 position == pos;
END pinionrack_BODY;
```

Figure 5-17 : Modèle VHDL-AMS de l'ensemble pignon-crémaillère.

La position du siège est calculée et limitée entre une position minimum et une position maximum (qui est atteinte "en douceur" grâce l'expression exponentielle du modèle précédent, figure 5.17).

La figure 5-18 présente l'élévation du siège en fonction de la tension appliquée au moteur.

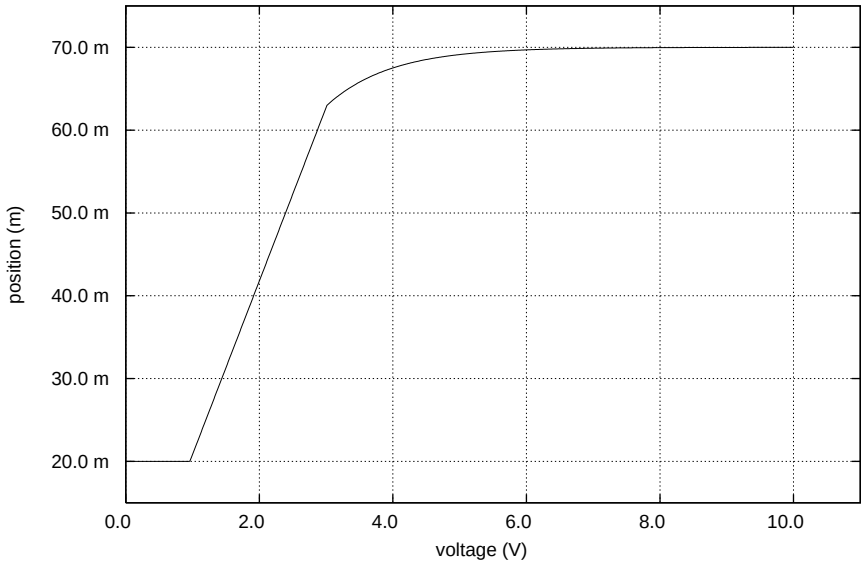


Figure 5-18 : Position du siège fonction de la tension appliquée au moteur.

5.4 Conclusion

Nous avons présenté dans ce chapitre deux applications multi-technologiques avec VamSpiceDesigner.

Dans la première partie, après avoir présenté et défini les éléments essentiels de la colorimétrie, nous avons appliqué cette méthode à un système colorimétrique basé sur l'analogie que nous avons trouvé entre la couleur et l'électricité. Le système colorimétrique considéré permet d'analyser, à partir de la schématique, le rendu de la couleur d'un objet coloré à travers le brouillard et à une certaine distance de l'objet. Pour comparer les couleurs rendues par ce système, nous avons développé un outil qui lit le fichier de sortie de SPICE (SPICERAW) et à partir de celui-ci renvoie les couleurs observées à plusieurs distance. Cet outil appelé **Colorimetric v1.03**.

Dans la deuxième partie, nous avons modélisé et simulé le comportement d'un système électro-mécanique tel que l'élévation d'un siège de voiture. Ce système comporte des éléments électro-mécaniques tels qu'un moteur à courant continu, des engrenages, des convertisseurs de mouvements et un module pignon-crémaillère.

Ce chapitre complète donc de manière originale les chapitres précédents sur les potentialités de la modélisation en multi-technologie de VHDL-AMS et l'utilisation de VamSpiceDesigner pour la mise en œuvre de la modélisation comportementale et hiérarchique et la simulation SPICE.

CONCLUSION GENERALE ET PERSPECTIVES

Les travaux que nous avons effectués au cours de cette thèse veulent apporter une contribution au développement de l'aide à la conception des systèmes multi-technologiques et mixtes.

L'étude, fait partie des travaux de recherche de l'équipe Modélisation Comportementale pour le Design au Département COMELEC de l'ENST. Elle a été initialisée par deux thèses de doctorat apportant également une contribution, l'une sur l'application du langage de modélisation et l'élaboration d'un outil de simulation, l'autre sur la méthodologie de conception de systèmes mixtes.

Cette thèse contribue à approfondir ces deux aspects : méthodes et outils de conception de systèmes complexes, portant d'avantage notre effort sur la conception de systèmes multi-technologiques.

Le contexte industriel européen a avantageusement favorisé le développement de l'étude, car nous avons participé activement à la réalisation du projet européen SPARTE. L'objectif de ce projet étant de proposer des méthodes et des outils permettant d'évaluer par modélisation et simulation les performances de composants électroniques commerciaux dans des gammes de températures étendues de [–50°C à +125°C] pour lesquelles ces composants ne sont pas spécifiés, permettant ainsi de limiter au juste besoin des tests systématiques très longs et coûteux. Ce contexte nous a guidé vers le choix du langage de modélisation, à savoir VHDL-AMS, l'élaboration de méthodes de modélisation de systèmes multi-technologiques (électro-thermique, colorimétrique, électro-mécanique) et l'outil d'aide à la conception de tels systèmes, à savoir **VamSpiceDesigner**.

VamSpiceDesigner apparaît dores et déjà comme un outil schématique d'aide à la conception de multi-technologique :

- En modélisation, nous pouvons utiliser ses potentialités hiérarchiques de navigation à différents niveaux d'abstraction aussi bien avec VHDL-AMS (modélisation fonctionnelle, comportementale et structurelle, et physique) qu'avec SPICE (macro-modélisation),
- En simulation, nous utilisons SPICE, le simulateur reconnu comme standard de fait de la modélisation.

Grâce à VamSpiceDesigner, l'outil de modélisation et de simulation, nous avons élaboré et validé des modèles de différents niveaux d'abstraction : fonctionnels, comportementaux et structurels, et physiques. Nous avons développé, au niveau schématique, une bibliothèque pour

arranger les modèles multi-technologiques. Cette bibliothèque a pour but la réutilisation des modèles développés. Avec les deux versions de cet outil, nous avons ainsi proposé une nouvelle méthodologie de conception multi-technologie (modélisation et simulation) basée sur une schématique hiérarchique.

Dans la deuxième partie, nous avons modélisé et simulé avec ADVanceMS des dispositifs MOS, à savoir le transistor MOS niveau 3 de SPICE, une cellule mémoire EEPROM, et un modèle de transistor MOS appelé UNICEL, au niveau physique. Par ces exemples, nous nous sommes mieux familiarisés avec la norme de VHDL-AMS et nous avons vu l'apport de VHDL 1076.1 par rapport au standard de fait SPICE.

Pour répondre à l'étude menée par le Projet Européen SPARTE, nous avons contribué à la modélisation et à la simulation électro-thermique des mémoires FLASH et SDRAM prises comme démonstrateurs dans le cadre de ce projet. L'étude concernant la mémoire FLASH est faite à l'aide de l'outil de simulation ADVanceMS de Mentor Graphics et celle relative à la mémoire SDRAM est complètement menée avec VamSpiceDesigner. Cette dernière est complétée par une étude comparative des résultats de mesure et de simulation en fonction de la température.

Dans le dernier chapitre, nous avons présenté les potentialités de la modélisation en multi-technologie de VHDL-AMS et l'utilisation de VamSpiceDesigner. Nous avons modélisé deux applications :

- Un modèle colorimétrique
- Un modèle électro-mécanique

Dans la première application, nous avons modélisé et simulé un système colorimétrique : il s'agit d'un panneau réfracteur de signalisation illuminé par une source de lumière. La lumière réfléchie se propage dans le milieu environnent, et nous étudions, au plan colorimétrique, la perception que nous avons de ce panneau en présence de brouillard. Afin comparer les couleurs rendues par ce système, nous avons développé un outil qui à partir de données SPICE détermine les couleurs rendues à plusieurs distances et densités de brouillard k. Cet outil est appelé ***Colorimetric v1.03***. Ce dernier est développé à partir des données standard de la CIE (Commission Internationale de L'Eclairage).

Dans la deuxième application, nous avons modélisé et simulé un système électro-mécanique : il s'agit d'une commande électrique permettant une variation de niveau vertical d'un siège d'automobile.

Les perspectives de développement en modélisation comportementale et en développement d'outils de simulation sont nombreuses. Citons-en quelques-unes qui nous apparaissent comme importantes :

- Le développement du numérique dans VamSpiceDesigner, qui permettrait d'étendre ses potentialités dans la conception de systèmes mixtes,
- L'intégration d'un compilateur C de GNU dans notre compilateur VHDL-AMS dont le but serait d'améliorer la compilation et les messages d'erreur.
- La restructuration et la finalisation de l'aide en ligne de l'outil VamSpiceDesigner
- L'association de la caractérisation « comportementale » comme complément indispensable de la modélisation « comportementale »,
- La possibilité d'effectuer des analyses de sensibilité des paramètres des modèles « comportementaux »,

Les deux derniers points utiliseraient avantageusement les potentialités d'optimisation de SPICE OPUS (intégré dans VamSpiceDesigner).

REFERENCES

ADV99 AHO00 AMS99 ANT88 BEN99 BOY74 CHA00 CHA98
CHA99 CON92 COX92 DAN79 FOT97 GUI31 HDL94 HER02
HUB97 IEE98 JEC03 JEM00 JEM03 JRL92 KUN95 LED93
LEM95 LOU97 MAN95 MEY71 MIL97 NAG75 NAN00 OAL98
OUD99 POR03 PUH98 QUA89 RAS96 RAS99 SHI68 SPA00
STI55 TUR83 VLA80 VLA94 VLAD94 WILL87 WRI28 WWW1
WWW2 WWW3 WWW4 WYS82 YAC96

[ADV99] : Mentor Graphics,
ADVance MS™, VHDL-AMS Design Station User's Manual,
Doc, Software Version v1.0_1.1, March 1999, Revision 1.0, DN : 303868(i). ↑

[AHO00] : A. Aho, R. Sethi, J. Ullman
Compilateurs, Principes, techniques et outils. – 900 pages
InterEditions, Paris, 1991 et Dunod, Paris, 2000. ↑

[AMS99] : Design Automation Standards Committee of the IEEE Computer Society
IEEE Standard VHDL Analog and Mixed Signal Extensions. – 314 pages
Doc., IEEE Std 1076.1-1999, 18 March 1999. ↑

[ANT88] : P. Antognetti, G. Massobrio,
Semiconductor device modeling with SPICE, – 398 pages,
McGraw-Hill Book Company, USA, 1988. ↑

[BEN99] : W. Benzarti,
Modélisation et caractérisation de la cellule mémoire de type EEPROM pour la simulation et la conception de circuits intégrés analogiques et mixtes
thèse, ENST Paris, décembre 1999. ↑

[BOY74] : G.R. Boyle, D.O. Pederson, B.M. Cohn, J.E. Solomon,
Macromodeling of integrated circuit operational amplifiers
Solid-State Circuits, IEEE Journal of Solid-State Circuit, Vol.: 9 Issue: 6 , Dec 1974, Page(s): 353 –364. ↑

[CHA00] : J.-J. Charlot,
VHDL-AMS for multi-technology,
MIXDES 2000, Gdynia, Pologne, Juin 2000. ↑

[CHA98] : J.-J. Charlot, E. Barker, O. Alali, J.-F. Charlot,
Color Rendering in a Hazy Environment : Simulation with SPICE3F5/VHDL-AMS,
BMAS'98, October 1998, Orlando, USA. ↑

[CHA99] : J.-J. Charlot, J.-K. Seon, J.-F. Charlot,
Analysis of colorimetric system under foggy, thermal and electrical conditions with SPICE/VHDL-AMS,
Elsevier Science Ltd, Microelectronics reliability, 2000. ↑

[CON92] : J. A. Connelly, Pyung Choi,
Macromodeling with SPICE, - 274 pages.
Englewood Cliffs, NJ : Prentice Hall, 1992. ↑

[COX92] : F.L. Cox, W.B. Kuhn, al.,
XSPICE, Software user's manual, 240 pages,
Computer Science and Information Technology Lab., Georgia Tech Research Institue, Atlanta, Georgia 30332. ↑

[DAN79] : L. Dang,
A simple current model for short-channel IGFET and its application to circuit simutation,
IEEE trans. On Election devices, Vol. ED-26, april. 1979. ↑

[FOT97] : D. P. Foty,
MOSFET modeling with SPICE, Principe and Practice, – 653 pages,
Prentice Hall PTR, New Jersey, USA, 1997. ↑

[GUI31] : J. Guild,
The colorimetric properties of the spectrum, page(s) 149 – 187.
Philosophical Transactions of the Royal Society of London,
A230, 1931. ↑

[HDL94] : ANACAD,
HDL-A User's Manual.
Doc., Revision v2.0, Software version v1.4_1, September 1994. ↑

[HER02] : Y. Hervé,
VHDL-AMS : Application et enjeux industriels,
cours et exercices corrigés. – 230 pages
DUNOD, Paris, 2002. ↑

[HUB97] : John Hubbard
Programmation en C++, Théorie et problème. – 430 pages
McGraw-Hill, Paris, 1997. ↑

[IEE98] : IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS),
Design Automation Standards committee of the IEEE computer socitety.
– 302 pages,
Doc., the Institute of Electrical and Electronics Enginers, April 17, 1998. ↑

[JEC03] : S.Jemmali, J.-J Charlot,
Colorimetric v1.03,
ENST Paris, France, Juin 2003. ↑

[JEM00] : S. Jemmali, J.-J. Charlot,
Apport de VHDL-AMS dans la modélisation et la simulation de circuits en technologie MOS,
JSFT'2000, Monastir, Tunisie, 2000. ↑

[JEM03] : S. Jemmali, A. Nehme, J.-J. Charlot,
Behavioral modeling of multi-technological systems
with VHDL-AMS and simulating with SPICE,
Some applications by using VamSpiceDesigner©,
BMAS'03, October 2003, San Jose, CA USA. ↑

[JRL92] : J.R. Levine, T. Mason, D. Brown
Lex & yacc. – 2e ed.
2e ed, O'Reilly & Associates, Inc., USA, 1990-1992. ↑

[KUN95] : K.S. Kundert,
The Designer's guide to SPICE and SPECTRE®, - 380 pages,
Kluwer Academic Publishers Boston USA, 1995. ↑

[LED93] : LEDA S.A.,
LEDA Procedure Interface (LPI), 315 pages,
Doc., LEDA, 1993-1997, Granier, France. ↑

[LEM95] : F. Lémery,
Modélisation comportementale des circuits analogiques et mixte, 282 pages,
Th., Institut National Polytechnque de Grenoble, 1995, France. ↑

[LOU97] : M. Loukides, A. Oram
GNU, - 255 pages
O'Reilly™, Paris, 1997. ↑

[MAN95] : H. A. Mantooth, M. Fiegenbaum,
Modeling with an Analog Hardware Description Language,
Kluwer Academic Publishers, 1995. ↑

[MEY71] : J.E. Meyer
MOS Models and circuit,
RCA Rev. Vol 32, Mars 1971, page(s) 42-63. ↑

[MIL97] : N. Milet-Lewis,
Contribution à la modélisation comportemental des circuits analogique,
155 pages,
Th., Université de Bordeaux 1, Novembre 1997, France. ↑

[NAG75] : L.W. Nagel,
SPICE2 : A computer program to simulate semi-conductor circuits. – 500 pages
Memorandum No. ERL-M520, Electronics Research Laboratory,
University of California, Berkeley, CA., USA, 9 May 1975. ↑

[NAN00] : R. Nanko,
Microsoft® Visual C++® 6.0, - 464 pages,
Micro Application, Paris, 2000. ↑

[OAL98] : O. ALALI,
Modélisation VHDL-AMS analogique et simulation SPICE, – 144 pages
Th. : Electronique et Communications : ENST Paris : 1998 ; 98E006. ↑

[OUD99] : J. Oudinot,
Méthodologie de Conception d'ASIC mixtes avec VHDL-AMS, – 144 pages
Th. : Electronique et Communications : ENST Paris : 1999 ; 2000E039. ↑

[POR03] : J. Porte,
Documentations OCEANE : Outils pour la Conception et l'Enseignement des C.I. ANalogiquE,
DOC., ENST Paris, 2003. ↑

[PUH98] : J. Puhan, T. Tuma, I. Fajfor,
SPICE for Windows 95/98/NT.
The University of Ljubljana, Slovenia, at the Faculty of Electrical Engineering,
1998. < http://fides.fe.uni-lj.si/spice/>. ↑

[QUA89] : Thomas L. Quarles,
SPICE3 Version 3C1 User Guide, - 80 pages
DOC., Electronics Research Lab., Berkeley, Memorandum No. UCB/ERL M89/46,
April 1989. ↑

[RAS96] : R. Rascalon, M. Romdhani, al.,
Méthode et langages pour la spécification des systèmes complexes, pages 69 à77,
REE, N° 3, Mars 1996. ↑

[RAS99] : R. Rascalon, J.-J. Charlot,
Conception des systèmes complexes : vers une utilisation accrue des interfaces graphiques, pages 6 à 12,
REE, N° 3, Mars 1999. ↑

[SHI68] : H. Shichman, D.A. Hodges,
Modeling and simulation of insulated-gate field-effect transistor switching Circuits,
IEEE Journal Solid-State Circuits, Volume: 3 Issue: 3 , Sep 1968, page(s):285 –289.
↑

[SPA00] : **SPARTE : Simulation based Performance Assessment & Rating regarding Thermal & Electrical effects**
EUCLID Contract RTP 2.34 N° 00/EF 2.34/004.
SPARTE Consortium : MBDA France, EPSILON Ingénierie, ENST Paris, SENER Ingenieria y Sistemas, CNM-IMB, et IXL Bordeaux I.

[STI55] : W.S. Stiles, J.M. Burch,
Interim report to the Commission International de l'Eclairage Zurich,
page(s) 168-181.
National Physical Laboratory's investigation of colour-matching,
Optica Acta, 2, 1955. ↑

[TUR83] : C. Turchetti, G. Masetti,
A macromodel for integrated all-MOS operational amplifiers.
Solid-State Circuits, IEEE Journal of Solid-State Circuit, Vol.: 18 Issue: 4 , Aug 1983, Page(s) : 389 – 394. ↑

[VLA80] : A. Vladimirescu, S. Liu,
The Simulation of MOS integrated circuits using SPICE 2,
Electron. Res. Lab., Univ. Of Calif., Berkeley, Memo ERM-M85/7, Oct. 1980. ↑

[VLA94] : A. Vladimirescu
The SPICE Book. – 413 pages
J.Wiley & Sons, Inc., NY, 1994. ↑

[VLAD94] : A. Vladimirescu, J.-J. Charlot
Mos analogue circuit simulation with SPICE,
IEE Proceedings circuit, devices and systems,
Vol. 141, No 4, August 1994, pp. 265-274. ↑

[WILL87] : P.J. Willis,
Visual simulation of atmospheric haze, page(s) 35 – 42.
Computer Graphics Forum, 1987. ↑

[WRI28] : W.D. Wright,
A re-determination of the trichromatic coefficients of the spectral colours,
page(s) 141-164.
Transactions of the Optical Society, 30, 1928. ↑

[WWW1] : eCircuitCenter,
http://www.ecircuitcenter.com/SPICEtopics.htm. ↑

[WWW2] : CIE (1931) 2-deg color matching functions
http://cvision.ucsd.edu/database/text/cmfs/ciexyz31.htm. ↑

[WWW3] : **CIE (International commission on illumination)**
http://www.cie.co.at/cie/**.** ↑

[WWW4] : **Where's purple ? Or, how to plot colours properly on a computer screen,**
http://casa.colorado.edu/ãjsh/colour/rainbow.html**.** ↑

[WYS82] : G. Wyszecki, W.S. Stiles,
Color Science : concepts and methods, quantitative data and formulae,
Wiley 2nd ed., New York, 1982. ↑

[YAC96] : Compiler Resources, Inc.
Yacc++ and the Language Objects Library Reference Guide.
Doc., Revision 2.1, July 1996. ↑

PUBLICATIONS SCIENTIFIQUES ET DEPOT DE COPYRIGHT

Publications Scientifiques

[JET03] : S.Jemmali, A.Rabhi, et J.-J.Charlot,
VHDL-AMS pour la colorimétrie,
une application utilisant VamSpiceDesigner,
JSF'03 (Journée Scientifique Francophone), Décembre 2003, Tozeur, Tunisie.

[JEMS03] : S. Jemmali, A.Rabhi, J.-J. Charlot,
Consumption of a SDRAM Memory Cell Module by using VamSpiceDesigner, a Design Tool based on VHDL-AMS and SPICE,
ICIT'03 (International Conference on Industrial Technology),
December 2003, Maribor, Slovenia.

[JEM03] : S. Jemmali, A. Nehme, J.-J. Charlot,
Behavioral modeling of multi-technological systems
with VHDL-AMS and simulating with SPICE,
Some applications by using VamSpiceDesigner,
BMAS'03, october 2003, San Jose, CA USA.

[JEP03] : S. Jemmali, J.-J. Charlot,
VamSpiceDesigner, a hierarchical schematic design tool of multi-technological systems based on VHDL-AMS and SPICE,
MIXDES 2003, Lodz, Pologne, juin 2003

[JECH03] : S.Jemmali, J.-J Charlot,
Colorimetric v1.03,
GET/ENST Paris, France, Juin 2003.

[JEF03] : S. Jemmali, J.-J. Charlot,
Détermination de la consommation d'une mémoire SDRAM à l'aide de VamSpiceDesigner basé sur VHDL-AMS et SPICE,
FTFC 2003, Paris, France, mai 2003.

[JES03] S. Jemmali, J.-J. Charlot,
VamSpiceDesigner, un outil de conception hiérarchique de systèmes multi-technologiques à base de VHDL-AMS et de SPICE,
SETIT 2003, Sousse, Tunisie, mars 2003.

[JEG02] S. Jemmali, J.-J. Charlot,
Vers une méthodologie de conception hiérarchique de systèmes multi-technologiques avec la contrainte de la température,
JNRDM 2002, Grenoble, France, 2002.

[JEW02] : S. Jemmali, J.-J. Charlot,
Towards a Hierarchical Multi-Technological Design Methodology Using VHDL-AMS,
MIXDES 2002, Wroclaw, Pologne, juin 2002.

[JES01] : S. Jemmali, J.-J. Charlot,
Le modèle UNICELL et l'apport de VHDL-AMS dans la modélisation et la simulation de circuits en technologie MOS,
JNRDM 2001, Strasbourg, France, 2001.

[JEF01] : S. Jemmali, J.-J. Charlot,
Vers une méthodologie de conception hiérarchique de systèmes multi-technologiques avec la contrainte de la température,
FTFC 2001, Paris, France, 2001.

[JEM00] : S. Jemmali, J.-J. Charlot,
Apport de VHDL-AMS dans la modélisation et la simulation de circuits en technologie MOS,
JSFT'2000, Monastir, France, 2000.

[JEM00] : S. Jemmali, J.-J. Charlot,
Apport de VHDL-AMS dans la modélisation et la simulation de circuits en technologie MOS,
JSFT'2000, Monastir, Tunisie, 2000.

Dépôt de Copyright de logiciels

[JEC02] : S. Jemmali, J.-J. Charlot,
VamSpiceDesigner©,
GET 2002, Paris, France, 2002.

[CHJE02] : J.-J. Charlot, S. Jemmali,
VamSpice©,
GET 2002, Paris, France, 2002.

Annexe 1

MODELE VHDL-AMS D'UN TRANSISTOR MOS NIVEAU 3 DE SPICE

```
LIBRARY IEEE; USE IEEE.math_REAL.all;
LIBRARY STD;  USE STD.standard.all;
USE work.mosdata.all; USE work.electrical_systems.all; USE work.thermal_systems.all;

ENTITY MOS_L3 IS
  GENERIC(
    -- instance parameters
    -- MOS Level 3 device
      L             :REAL:= defaultMosL;            --Longueur de canal (m)
      W             :REAL:= defaultMosW;            --Largeur de canal (m)
      AS            :REAL:= 0.0;                    --Surface du source (m2)
      AD            :REAL:= 0.0;                    --Surface du drain (m2)
      PS            :REAL:= UNDEF;                  --Périmètre du source (m)
      PD            :REAL:= UNDEF;                  --Périmètre du drain (m)
      RDC           :REAL:= 0.0;                    --Résistance du drain due au résistivité du contact (Ohm)
      RSC           :REAL:= 0.0;                  --Résistance de la source due au résistivité du contact (Ohm)
      XA     :REAL:= defaultDiffusionLength;        --Longueur de diffusion (m)
      -- MOS Level 3 Model
      VTO           :REAL:= UNDEF;                  --Tension de seuil à polarisation de substrat nulle (Volt)
      KP            :REAL:= 2.0E-5;                 --La transconductance (A/Volt2)
      GAMMA         :REAL:= 0.0;      --Effet de la polarisation de substrat sur la tension de seuil (Volt1/2)
      PHI           :REAL:= UNDEF;     --Potentiel de la surface dans la région de forte inversion (Volt)
      RD            :REAL:= UNDEF;                  --Résistance du drain (Ohm)
      RS            :REAL:= UNDEF;                  --Résistance de la source (Ohm)
      CBD           :REAL:= UNDEF;                  --Capacité de jonction B-D à polarisation nulle (F)
      CBS           :REAL:= UNDEF;                  --Capacité de jonction B-S à polarisation nulle (F)
      IS_o          :REAL:= 1.0E-14;                --Courant de saturation des diodes (A)
      PB            :REAL:= 0.8;                    --Potentiel de la jonction (Volt)
      CGSO          :REAL:= 0.0;                    --Capacité de débordement G-S (F)
      CGDO          :REAL:= 0.0;                    --Capacité de débordement G-D (F)
      CGBO          :REAL:= 0.0;                    --Capacité de débordement G-B (F)
      RSH           :REAL:= UNDEF;                  --Résistance/carrée des diffusions de S et D (Ohm/□)
      CJ            :REAL:= UNDEF;                  --Capacité / surface de la jonction de B (de fond) (F/m2)
      MJ            :REAL:= 0.5;                    --Coeff pour la Capacité de la jonction de B (de fond) (-)
      CJSW          :REAL:= UNDEF;                  --Capacité / mètre de la jonction de B (de coté) (F/m)
      MJSW          :REAL:= 0.33;                   --Coeff pour la Capacité de la jonction de B (de coté) (-)
      JS            :REAL:= 0.0;                    --Densité de courant de saturation des diodes (A/m2)
      TOX           :REAL:= 1.0E-7;                 --Epaisseur de l'oxyde sous la grille (m)
      LD            :REAL:= 0.0;                    --Diffusion latérale (m)
      U0            :REAL:= UNDEF;                  --Mobilité des porteurs (cm2/(V.s))
```

```
    FC          :REAL:= 0.5;
    NSUB        :REAL:= UNDEF;              --Dopage du substrat (AT/cm3)
    TPG         :REAL:= UNDEF;              --Type de matériel de la grille (-)
    NSS         :REAL:= UNDEF;              --Densité d'état de surface (AT/cm2)
    ETA         :REAL:= 0.0;                --Effet statique (-)
    DELTA       :REAL:= 0.0;                --Effet de largeur de canal sur la tension de seuil (1/Volt)
    NFS         :REAL:= 0.0;                --Densité d'état de surface rapide (AT/cm2)
    THETA       :REAL:= 0.0;                --Modulation de la mobilité (1/Volt)
    VMAX        :REAL:= 0.0;              --Limitation vitesse porteurs champs électriques importants (m/s)
    KAPPA       :REAL:= 0.2;                --Facteur de champ de saturation (1/Volt)
    XJ          :REAL:= 0.0;                --Profondeur de la diffusion (m)
    UEXP        :REAL:= UNDEF;            --Coeff. pour la mobilité pour un champ critique (exposant) (-)
    NEFF        :REAL:= UNDEF;            --Coeff. pour la charge totale (mobile et fixe) dans le canal (-)
    XD          :REAL:= UNDEF;
    ALPHA       :REAL:= UNDEF;              --Coeff. de courent pour l'ionisation par impact (1/Volt)
    KF          :REAL:= 0.0;                --Flicker noise coeff. (-)
    AF          :REAL:= 1.0;                --Flicker noise exponent (-)
    BEX         :REAL:= -1.5;               --Coeff. de température pour mobilité
    TRD1        :REAL:=0.0;                 --Coeff. de température (linaire) RD (1/°K)
    TRD2        :REAL:=0.0;                 --Coeff. de température (quadratique) RD (1/°K2)
    TRS1        :REAL:=0.0;                 --Coeff. de température (linaire) RS (1/°K)
    TRS2        :REAL:=0.0;                 --Coeff. de température (quadratique) RS  (1/°K2)
    TYPE_TR     :REAL:= NMOS;               -- Type de transistor (NMOS, PMOS)
    TNOM        :REAL:= UNDEF               --Température nominale (°C)
    );
  PORT( TERMINAL  drain,  gate,  source,  bulk  : electrical
        --TERMINAL T1 :thermal
    );
END ENTITY MOS_L3;

ARCHITECTURE mos3 OF MOS_L3 IS
--*****************************************************************************
FUNCTION max(a,b:REAL) return REAL IS
BEGIN
  IF (a>b) THEN return a; ELSE return b; END IF;
END FUNCTION;
--*****************************************************************************
FUNCTION min(a,b:REAL) return REAL IS
BEGIN
  IF (a>b) THEN return b; ELSE return a; END IF;
END FUNCTION;
--*****************************************************************************
FUNCTION MEYER_C_CGB(Vgs,Vgd,Von:VOLTAGE; phi_1,cox_1:REAL; Leff:REAL
                     ) return CAPACITANCE IS

VARIABLE Cgb, GateBulkOverlapCap :CAPACITANCE;
VARIABLE Vds :VOLTAGE;
VARIABLE  a, b:REAL;
VARIABLE model_MOS3gateBulkOverlapCapFactor :REAL:= CGBO;
BEGIN
GateBulkOverlapCap := model_MOS3gateBulkOverlapCapFactor * Leff;
  Vds := Vgs-Vgd;
  --Calcule de la capacite de CGB
  IF (Vgs < (Von - phi_1)) THEN                                    -- Accumulqtion region
    Cgb := cox_1 + GateBulkOverlapCap;
  ELSIF (Vgs > (Von - phi_1)) and (Vgs < Von) THEN                 -- Depletion region
    Cgb := cox_1 *(1.0 +((2.0/3.0)*(Von - phi_1 - Vgs)/(phi_1))) + GateBulkOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN                   -- Saturation region
    Cgb := GateBulkOverlapCap;
  ELSE                                                             -- Linear region
```

```
    Cgb := GateBulkOverlapCap;
  END IF;
return Cgb;
END FUNCTION;
--*****************************************************************************
FUNCTION MEYER_C_CGS(Vgs,Vgd,Von:VOLTAGE;
                     phi_1, cox_1:REAL;
                     Weff:REAL
                    ) return CAPACITANCE IS
VARIABLE Cgs, GateSourceOverlapCap :CAPACITANCE;
VARIABLE Vds :VOLTAGE;
VARIABLE model_MOS3gateSourceOverlapCapFactor :REAL:= CGSO;
BEGIN
  GateSourceOverlapCap := model_MOS3gateSourceOverlapCapFactor * Weff;
  Vds := Vgs - Vgd;
  --Calcule de la capacite de CGS
  IF (Vgs < Von )  THEN                                                    -- Accumulqtion region
    Cgs := GateSourceOverlapCap;
  ELSIF  (Vgs = Von) THEN     -- Depletion region
    Cgs := 2.0/3.0*cox_1*((Von - Vgs)/phi_1 + 1.0) + GateSourceOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN                           -- Saturation region
    Cgs := 2.0/3.0*cox_1 + GateSourceOverlapCap ;
  ELSE                                                                     -- Linear region
    Cgs := 2.0/3.0*cox_1*(1.0 - ( (Vgs - Vds - Von)/(2.0*(Vgs - Von) - Vds) )**2) + GateSourceOverlapCap;
  END IF;
return Cgs;
END FUNCTION;
--*****************************************************************************
FUNCTION MEYER_C_CGD(Vgs,Vgd,Von:VOLTAGE; phi_1, cox_1:REAL; Weff:REAL
                                    ) return CAPACITANCE IS

VARIABLE GateDrainOverlapCap, Cgd :CAPACITANCE;
VARIABLE Vds :VOLTAGE;
VARIABLE model_MOS3gateDrainOverlapCapFactor :REAL:= CGDO;
BEGIN
  GateDrainOverlapCap := model_MOS3gateDrainOverlapCapFactor * Weff;
  Vds := Vgs - Vgd;
  -- Calcule de la capacite de CGD
  IF (Vgs < (Von - phi_1)) THEN                                            --Accumulqtion region
    Cgd := GateDrainOverlapCap;
  ELSIF (Vgs > (Von - phi_1)) and (Vgs < Von) THEN                         --Depletion region
    Cgd := GateDrainOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN                           --Saturation region
    Cgd := GateDrainOverlapCap;
  ELSE                                                                     --Linear region
    Cgd := 2.0/3.0*cox_1*(1.0 - ( (Vgs - Von)/(2.0*(Vgs - Von) - Vds) )**2) + GateDrainOverlapCap;
  END IF;
return Cgd;
END FUNCTION;
--*****************************************************************************
PROCEDURE EQU_Model_MOS3(Vdsq, Vgsq, Vbsq          :IN REAL;
                         model_MOS3modName,here_MOS3name   :IN STRING;
                         MOS3_TEMP                 :IN REAL;
                         MOS3_Id,MOS3_Is,MOS3_Ib   :OUT CURRENT;
                         MOS3_gm, MOS3_gds, MOS3_gmbs      :OUT REAL;
                         MOS3_cgd, MOS3_cgs, MOS3_cgb      :OUT CAPACITANCE;
                         MOS3_capbd, MOS3_capbs            :OUT CAPACITANCE
                        ) IS

--*****************************
```

```
--Les autres parametres
--*****************************
VARIABLE model_MOS3vt0 :REAL:= VTO;
VARIABLE model_MOS3transconductance :REAL:= KP;
VARIABLE model_MOS3gamma :REAL:= GAMMA;
VARIABLE model_MOS3phi :REAL:= PHI;
VARIABLE model_MOS3capBD :REAL:= CBD;
VARIABLE model_MOS3capBS :REAL:= CBS;
VARIABLE model_MOS3jctSatCur :REAL:= IS_o;
VARIABLE model_MOS3bulkJctPotential :REAL:= PB;
VARIABLE model_MOS3bulkCapFactor :REAL:= CJ;
VARIABLE model_MOS3bulkJctBotGradingCoeff :REAL:= MJ;
VARIABLE model_MOS3sideWallCapFactor :REAL:= CJSW;
VARIABLE model_MOS3bulkJctSideGradingCoeff :REAL:= MJSW;
VARIABLE model_MOS3jctSatCurDensity :REAL:= JS;
VARIABLE model_MOS3oxideThickness :REAL:= TOX;
VARIABLE model_MOS3latDIFf :REAL:= LD;
VARIABLE model_MOS3surfaceMobility :REAL:= U0;
VARIABLE model_MOS3MobilityTempCoeff :REAL:=BEX;
VARIABLE model_MOS3fwdCapDepCoeff :REAL:= FC;
VARIABLE model_MOS3substrateDoping :REAL:= NSUB;
VARIABLE model_MOS3gateType :REAL:= TPG;
VARIABLE model_MOS3surfaceStateDensity :REAL:= NSS;
VARIABLE model_MOS3eta :REAL:= ETA;
VARIABLE model_MOS3delta :REAL:= DELTA;
VARIABLE model_MOS3fastSurfaceStateDensity :REAL:= NFS;
VARIABLE model_MOS3theta :REAL:= THETA;
VARIABLE model_MOS3maxDrIFtVel :REAL:= VMAX;
VARIABLE model_MOS3kappa :REAL:= KAPPA;
VARIABLE model_MOS3junctionDepth :REAL:= XJ;
VARIABLE model_MOS3tnom :REAL:= TNOM;
VARIABLE model_MOS3fNcoef :REAL:= KF;
VARIABLE model_MOS3fNexp :REAL:= AF;
VARIABLE model_MOS3type  :REAL:= TYPE_TR;
VARIABLE model_MOS3oxideCapFactor :REAL;
VARIABLE model_MOS3narrowFactor :REAL;
VARIABLE model_MOS3alpha :REAL;
VARIABLE model_MOS3coeffDepLayWidth:REAL;
--*************************************************
-- Device model
--*************************************************
VARIABLE here_MOS3w :REAL:= W;
VARIABLE here_MOS3l :REAL:= L;
VARIABLE here_MOS3sourceArea :REAL :=AS;
VARIABLE here_MOS3drainArea :REAL:=AD;
VARIABLE here_MOS3drainPerimiter :REAL:=PD;
VARIABLE here_MOS3sourcePerimiter :REAL:=PS;
VARIABLE here_MOS3DiffusionLength :REAL := XA;

VARIABLE here_MOS3temp :REAL:= MOS3_TEMP;
VARIABLE here_MOS3vdsat :REAL:= 0.0;
VARIABLE here_MOS3von :REAL:= 0.0;
VARIABLE here_MOS3mode :REAL;
VARIABLE here_MOS3sourceConductance:REAL;
VARIABLE here_MOS3drainConductance :REAL;
VARIABLE here_MOS3tPhi:REAL;
VARIABLE here_MOS3tSurfMob:REAL;
VARIABLE here_MOS3tTransconductance:REAL;
VARIABLE here_MOS3tBulkPot:REAL;
VARIABLE here_MOS3tCjsw_C:REAL;
```

```
VARIABLE here_MOS3tCbd_C:REAL;
VARIABLE here_MOS3capbd :CAPACITANCE;
VARIABLE here_MOS3capbs :CAPACITANCE;
VARIABLE here_MOS3capgb :CAPACITANCE;
VARIABLE here_MOS3capgd :CAPACITANCE;
VARIABLE here_MOS3capgs :CAPACITANCE;
VARIABLE here_MOS3cgd   :CAPACITANCE;
VARIABLE here_MOS3cgs   :CAPACITANCE;
VARIABLE here_MOS3cgb   :CAPACITANCE;
VARIABLE here_MOS3tCj_C:REAL;
VARIABLE here_MOS3tDepCap:REAL;
VARIABLE here_MOS3sourceVcrit:REAL;
VARIABLE here_MOS3drainVcrit:REAL;
VARIABLE here_MOS3tCbs_C:REAL;
VARIABLE here_MOS3Cbd_C:REAL;
VARIABLE here_MOS3Cbs_C:REAL;
VARIABLE here_MOS3Cbdsw_C:REAL;
VARIABLE here_MOS3f2d:REAL;
VARIABLE here_MOS3f3d:REAL;
VARIABLE here_MOS3f2s:REAL;
VARIABLE here_MOS3f3s:REAL;
VARIABLE here_MOS3Cbssw_C:REAL;
VARIABLE here_MOS3tSatCurDens:REAL;
VARIABLE here_MOS3tSatCur:REAL;
VARIABLE here_MOS3tVbi:REAL;
VARIABLE here_MOS3tVto:REAL;
VARIABLE here_MOS3cbs_i:REAL;
VARIABLE here_MOS3gbs:REAL;
VARIABLE here_MOS3gbd:REAL;
VARIABLE here_MOS3cbd_i:REAL;
VARIABLE here_MOS3gm :REAL:=0.0;
VARIABLE here_MOS3gds:REAL:=0.0;
VARIABLE here_MOS3gmbs :REAL:=0.0;
VARIABLE here_MOS3cd:REAL:=0.0;
VARIABLE egfet1, arg1, kt1, pbfact1, vtnom, fact1, fermis, fermig, wkfng, wkfngs, vfb, arg, pbfact,
        phio, fact2, ratio4, ratio5, egfet, kt, capfact, pbo, gmaold, gmanew, czbd, czbdsw, sarg,
        sargsw, czbs, czbssw, EffectiveLength,DrainSatCur, SourceSatCur, argsw, Beta, OxideCap,
        vbs, vgs, vgd, vbd, vds, vgb,  vt, ratio, evbs, evbd,  sqphis, vb, phibs, sqphbs, dsqdvb,wps,
        oneoverxj, xjonxl, oneoverxl, djonxj, wponxj, argb, dadvb, dbdvb, argc, dwpdvb, dfsdvb,
        fbody, gammas, fshort, vth, von, dqbdvb, dvtdvb, sqphs3, wconxj, arga, fbodys, dfbdvb,
        qbonco, vbix, dvtdvd, cdonco, xn, csonco, fgate, vgsx, onfg, vg, onfbdy, eta, dxndvb, dvodvd,
        cdrain, vdsat, dvodvb, us, dfgdvg, dfgdvd, dvsdvg, onvdsc, vdsc, ebd, dfgdvb, dvsdvd, dvsdga,
        cdnorm, dcodvb, vdsx, cdo, dvsdvb, cd1, fdrain, fd2, dfddvg, wfact, gms, gmw, delxl, dldvd,
        ddldvg, onxn, ondvt, gds0, xlfact, cdsat, gdsat, ddldvb, gdoncd, gdonfg, dgdvb, gdonfd, dfddvd,
        emax, demdvg, emoncd, dgdvg, dldem, emongd, ddldvd, dlonxl, dfddvb, dgdvd, demdvd,
        diddl, demdvb    :REAL;
BEGIN
IF(model_MOS3tnom = UNDEF) THEN
  model_MOS3tnom := defaulttnom + CONSTCtoK;
ELSE
  model_MOS3tnom := model_MOS3tnom + CONSTCtoK;
END IF;
fact1 := model_MOS3tnom/REFTEMP;
vtnom := model_MOS3tnom*CONSTKoverQ;
kt1 := CONSTboltz * model_MOS3tnom;
egfet1 := 1.16-(7.02e-4*model_MOS3tnom*model_MOS3tnom)/(model_MOS3tnom+1108.0);
arg1 := -egfet1/(kt1+kt1)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
pbfact1 := -2.0*vtnom *(1.5*log(fact1)+q*arg1);
model_MOS3oxideCapFactor := 3.9 * 8.854214871e-12/model_MOS3oxideThickness;
IF(model_MOS3surfaceMobility = UNDEF) THEN model_MOS3surfaceMobility:=600.0; END IF;
```

```
IF(model_MOS3transconductance = UNDEF) THEN model_MOS3transconductance :=
model_MOS3surfaceMobility * model_MOS3oxideCapFactor * 1.0e-4; END IF;
IF( model_MOS3substrateDoping/= UNDEF) THEN
  IF(model_MOS3substrateDoping*1.0e6 > 1.45e16) THEN --/*(cm**3/m**3)*/
    IF(model_MOS3phi = UNDEF) THEN
      model_MOS3phi := 2.0*vtnom*log(model_MOS3substrateDoping*1.0e6/1.45e16);--(cm**3/m**3)
      model_MOS3phi := max(0.1,model_MOS3phi);
    END IF;
    fermis := model_MOS3type * 0.5 * model_MOS3phi;
    wkfng := 3.2;
    IF(model_MOS3gateType = UNDEF) THEN model_MOS3gateType:=1.0; END IF;
    IF(model_MOS3gateType /= 0.0) THEN
      fermig := model_MOS3type * model_MOS3gateType*0.5*egfet1;
      wkfng := 3.25 + 0.5 * egfet1 - fermig;
    END IF;
    wkfngs := wkfng - (3.25 + 0.5 * egfet1 + fermis);
    IF(model_MOS3gamma = UNDEF) THEN
      model_MOS3gamma := sqrt(2.0 * EPSSIL * q * model_MOS3substrateDoping*1.0e6)/
       model_MOS3oxideCapFactor; --/*(cm**3/m**3)*/
    END IF;
    IF(model_MOS3vt0 = UNDEF) THEN
      IF(model_MOS3surfaceStateDensity = UNDEF) THEN
        model_MOS3surfaceStateDensity:=0.0; --NSS
        vfb := wkfngs - model_MOS3surfaceStateDensity * 1.0e4 * q/model_MOS3oxideCapFactor;
        model_MOS3vt0 := vfb + model_MOS3type *(model_MOS3gamma *
       sqrt(model_MOS3phi)+model_MOS3phi);
      ELSE
        vfb := model_MOS3vt0 - model_MOS3type *
       (model_MOS3gamma*sqrt(model_MOS3phi)+model_MOS3phi);
      END IF;
    END IF;
    model_MOS3alpha := (EPSSIL+EPSSIL)/(q*model_MOS3substrateDoping*1.0e6);-- /*(cm**3/m**3)*/ )
    model_MOS3coeffDepLayWidth := sqrt(model_MOS3alpha);
  ELSE
    model_MOS3substrateDoping := 0.0;
   report "ERROR FATAL Nsub < Ni" & model_MOS3modName
     severity ERROR; --WARNING;
  END IF;
END IF;
--***********************************
-- now model parameter preprocessing
--***********************************
model_MOS3narrowFactor := model_MOS3delta * 0.5 * M_PI * EPSSIL / model_MOS3oxideCapFactor;
--* perform the parameter defaulting *
IF(here_MOS3temp = UNDEF) THEN here_MOS3temp := defaulttemp + CONSTCtoK; END IF;
vt := here_MOS3temp * CONSTKoverQ;
ratio := here_MOS3temp/model_MOS3tnom;
fact2 := here_MOS3temp/REFTEMP;
kt := here_MOS3temp * CONSTboltz;
egfet := 1.16-(7.02e-4*here_MOS3temp*here_MOS3temp)/(here_MOS3temp+1108.0);
arg := -egfet/(kt+kt)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
pbfact := -2.0*vt *(1.5*log(fact2)+q*arg);
 IF((here_MOS3l - (2.0 * model_MOS3latDIFf)) <= 0.0) THEN
   report "Effective channel length less than zero : See LD or L" & here_MOS3name
     severity ERROR; --(WARNING);
 END IF;
 ratio4 := ratio * sqrt(ratio);
 here_MOS3tTransconductance := model_MOS3transconductance / ratio4;
 here_MOS3tSurfMob := model_MOS3surfaceMobility * ratio**model_MOS3MobilityTempCoeff;
 phio := (model_MOS3phi-pbfact1)/fact1;
```

```
here_MOS3tPhi := fact2 * phio + pbfact;
here_MOS3tVbi :=  model_MOS3vt0 - model_MOS3type *  (model_MOS3gamma*sqrt(model_MOS3phi))
+0.5*(egfet1-egfet)  + model_MOS3type*0.5*(here_MOS3tPhi-model_MOS3phi);
here_MOS3tVto := here_MOS3tVbi + model_MOS3type *  model_MOS3gamma * sqrt(here_MOS3tPhi);
here_MOS3tSatCur := model_MOS3jctSatCur * exp(-egfet/vt+egfet1/vtnom);
here_MOS3tSatCurDens := model_MOS3jctSatCurDensity * exp(-egfet/vt+egfet1/vtnom);

pbo := (model_MOS3bulkJctPotential - pbfact1)/fact1;
gmaold := (model_MOS3bulkJctPotential-pbo)/pbo;
capfact := 1.0/(1.0+model_MOS3bulkJctBotGradingCoeff * (4.0e-4*(model_MOS3tnom-REFTEMP)-
gmaold));
here_MOS3tCbd_C := model_MOS3capBD * capfact;
here_MOS3tCbs_C := model_MOS3capBS * capfact;
here_MOS3tCj_C := model_MOS3bulkCapFactor * capfact;
capfact := 1.0/(1.0+model_MOS3bulkJctSideGradingCoeff * (4.0e-4*(model_MOS3tnom-REFTEMP)-
gmaold));
here_MOS3tCjsw_C := model_MOS3sideWallCapFactor * capfact;
here_MOS3tBulkPot := fact2 * pbo+pbfact;
gmanew := (here_MOS3tBulkPot-pbo)/pbo;
capfact := (1.0+model_MOS3bulkJctBotGradingCoeff * (4.0e-4*(here_MOS3temp-REFTEMP)-gmanew));
here_MOS3tCbd_C := here_MOS3tCbd_C * capfact;
here_MOS3tCbs_C := here_MOS3tCbs_C * capfact;
here_MOS3tCj_C  := here_MOS3tCj_C * capfact;
capfact := (1.0+model_MOS3bulkJctSideGradingCoeff * (4.0e-4*(here_MOS3temp-REFTEMP)-
gmanew));
here_MOS3tCjsw_C := here_MOS3tCjsw_C * capfact;
here_MOS3tDepCap := model_MOS3fwdCapDepCoeff * here_MOS3tBulkPot;
IF( (model_MOS3jctSatCurDensity = 0.0) or (here_MOS3drainArea = 0.0) or (here_MOS3sourceArea =
0.0) ) THEN
   here_MOS3sourceVcrit := vt*log(vt/(CONSTroot2*model_MOS3jctSatCur));
   here_MOS3drainVcrit := vt*log(vt/(CONSTroot2*model_MOS3jctSatCur));
ELSE
   here_MOS3drainVcrit :=vt * log( vt / (CONSTroot2 * model_MOS3jctSatCurDensity *
here_MOS3drainArea));
   here_MOS3sourceVcrit := vt * log( vt / (CONSTroot2 * model_MOS3jctSatCurDensity *
here_MOS3sourceArea));
END IF;
-- Area Drain and Source
IF (here_MOS3sourceArea = UNDEF) THEN
  IF (here_MOS3w = defaultMosW ) THEN
    here_MOS3sourceArea := defaultMosAS;
  ELSE
    here_MOS3sourceArea := here_MOS3w * here_MOS3DiffusionLength;
  END IF;
END IF;
IF (here_MOS3drainArea = UNDEF) THEN
  IF (here_MOS3w = defaultMosW ) THEN
    here_MOS3drainArea  := defaultMosAD;
  ELSE
    here_MOS3drainArea := here_MOS3w * here_MOS3DiffusionLength;
  END IF;
END IF;
-- Permitier Drain and Source
IF (here_MOS3drainPerimiter  = UNDEF) THEN
  IF (here_MOS3w = defaultMosW ) THEN
    here_MOS3drainPerimiter  := defaultMosPD;
  ELSE
    here_MOS3drainPerimiter := here_MOS3w + 2.0 * here_MOS3DiffusionLength;
  END IF;
END IF;
```

```
 IF ( here_MOS3sourcePerimiter = UNDEF) THEN
   IF (here_MOS3w = defaultMosW ) THEN
      here_MOS3sourcePerimiter := defaultMosPS;
   ELSE
     here_MOS3sourcePerimiter:= here_MOS3w + 2.0 * here_MOS3DiffusionLength;
   END IF;
 END IF;

 IF(model_MOS3capBD /= UNDEF) THEN
    czbd := here_MOS3tCbd_C;
 ELSE
    IF(model_MOS3bulkCapFactor /= UNDEF) THEN
       czbd := here_MOS3tCj_C*here_MOS3drainArea;
    ELSE
       czbd := 0.0;
    END IF;
 END IF;
 IF(model_MOS3sideWallCapFactor /= UNDEF) THEN
    czbdsw := here_MOS3tCjsw_C * here_MOS3drainPerimiter;
 ELSE
    czbdsw := 0.0;
 END IF;
 arg := 1.0 - model_MOS3fwdCapDepCoeff;
 sarg :=   arg**(1.0 + model_MOS3bulkJctBotGradingCoeff);
 sargsw := arg**(1.0 + model_MOS3bulkJctSideGradingCoeff);
 here_MOS3Cbd_C := czbd;
 here_MOS3Cbdsw_C := czbdsw;
 here_MOS3f2d := czbd   *(1.0-model_MOS3fwdCapDepCoeff * (1.0 +
model_MOS3bulkJctBotGradingCoeff)) / sarg +
              czbdsw *(1.0-model_MOS3fwdCapDepCoeff * (1.0 + model_MOS3bulkJctSideGradingCoeff))/
sargsw;
 here_MOS3f3d := czbd * model_MOS3bulkJctBotGradingCoeff / (model_MOS3bulkJctPotential * sarg) +
              czbdsw * model_MOS3bulkJctSideGradingCoeff / (model_MOS3bulkJctPotential * sargsw);

 IF(model_MOS3capBS /= UNDEF ) THEN
    czbs := here_MOS3tCbs_C;
 ELSE
    IF(model_MOS3bulkCapFactor /= UNDEF) THEN
       czbs := here_MOS3tCj_C * here_MOS3sourceArea;
    ELSE
       czbs := 0.0;
    END IF;
 END IF;
 IF(model_MOS3sideWallCapFactor /= UNDEF) THEN
    czbssw := here_MOS3tCjsw_C * here_MOS3sourcePerimiter;
 ELSE
    czbssw := 0.0;
 END IF;

 arg := 1.0 - model_MOS3fwdCapDepCoeff;
 sarg := arg**(1.0 + model_MOS3bulkJctBotGradingCoeff);
 sargsw := arg**(1.0 + model_MOS3bulkJctSideGradingCoeff);
 here_MOS3Cbs_C := czbs;
 here_MOS3Cbssw_C := czbssw;
 here_MOS3f2s := czbs   * (1.0-model_MOS3fwdCapDepCoeff *
(1.0+model_MOS3bulkJctBotGradingCoeff))  / sarg +
              czbssw * (1.0-model_MOS3fwdCapDepCoeff * (1.0+model_MOS3bulkJctSideGradingCoeff)) /
sargsw;
 here_MOS3f3s := czbs * model_MOS3bulkJctBotGradingCoeff / (model_MOS3bulkJctPotential * sarg) +
              czbssw * model_MOS3bulkJctSideGradingCoeff  / (model_MOS3bulkJctPotential * argsw);
```

```
--************************Fin dependance en temperature*******************************

EffectiveLength := here_MOS3l - 2.0 * model_MOS3latDIFf;
IF( (here_MOS3tSatCurDens = 0.0) or (here_MOS3drainArea = 0.0) or (here_MOS3sourceArea = 0.0))
THEN
  DrainSatCur := here_MOS3tSatCur;
  SourceSatCur := here_MOS3tSatCur;
ELSE
  DrainSatCur := here_MOS3tSatCurDens * here_MOS3drainArea;
  SourceSatCur := here_MOS3tSatCurDens * here_MOS3sourceArea;
END IF;
Beta := here_MOS3tTransconductance * here_MOS3w/EffectiveLength;
OxideCap := model_MOS3oxideCapFactor * EffectiveLength * here_MOS3w;  --COX'

--*****************************************************************************
vbs := model_MOS3type * Vbsq; -- l'entree c'est Vbsq
vgs := model_MOS3type * Vgsq;
vds := model_MOS3type * Vdsq;
--vds := Vdsq;
vbd := vbs - vds;
vgd := vgs - vds;
vgb := vgs - vbs;

--* bulk-source and bulk-drain diodes
IF(vbs <= 0.0) THEN
  here_MOS3gbs := SourceSatCur/vt;
  here_MOS3cbs_i := here_MOS3gbs*vbs;  --Bulk-Source Current
  here_MOS3gbs :=  here_MOS3gbs + ckt_CKTgmin;
ELSE
  evbs := exp(MIN(MAX_EXP_ARG,vbs/vt));
  here_MOS3gbs := SourceSatCur*evbs/vt + ckt_CKTgmin;
  here_MOS3cbs_i := SourceSatCur * (evbs-1.0);
END IF;
IF(vbd <= 0.0) THEN
  here_MOS3gbd := DrainSatCur/vt;
  here_MOS3cbd_i := here_MOS3gbd *vbd; --Bulk-Drain Current
  here_MOS3gbd := here_MOS3gbd + ckt_CKTgmin;
ELSE
  evbd := exp(MIN(MAX_EXP_ARG,vbd/vt));
  here_MOS3gbd := DrainSatCur*evbd/vt +ckt_CKTgmin;
  here_MOS3cbd_i := DrainSatCur *(evbd-1.0);
END IF;

--* now to determine whether the user was able to correctly
--* identify the source and drain of his device
--*
IF(vds >= 0.0) THEN
   --* normal mode *
   here_MOS3mode := 1.0;
ELSE
   --* inverse mode *
   here_MOS3mode := -1.0;
END IF;
--*****************************************************
--*     reference cdrain equations to source and
--*     charge equations to bulk
--*****************************************************
vdsat := 0.0;
oneoverxl := 1.0/EffectiveLength;
```

```
eta := model_MOS3eta * 8.15e-
22/(model_MOS3oxideCapFactor*EffectiveLength*EffectiveLength*EffectiveLength);
--*******************************
--* square root term
--*******************************
IF (here_MOS3mode = 1.0) THEN
   vb := vbs;
ELSE
   vb := vbd;
END IF;

IF ( vb <=  0.0 ) THEN
   phibs  :=  here_MOS3tPhi - vb;
   sqphbs  :=  sqrt(phibs);
   dsqdvb  :=  -0.5/sqphbs;
ELSE
   sqphis := sqrt(here_MOS3tPhi);
   sqphs3 := here_MOS3tPhi*sqphis;
   sqphbs := sqphis/(1.0+ vb/(here_MOS3tPhi+here_MOS3tPhi));
   phibs := sqphbs*sqphbs;
   dsqdvb := -phibs/(sqphs3+sqphs3);
END IF;
--*********************************
--*short channel effect factor
--*********************************
IF ((model_MOS3junctionDepth /= 0.0) and (model_MOS3coeffDepLayWidth /= 0.0) ) THEN
   wps := model_MOS3coeffDepLayWidth*sqphbs;
   oneoverxj := 1.0/model_MOS3junctionDepth;
   xjonxl := model_MOS3junctionDepth*oneoverxl;
   djonxj := model_MOS3latDIFf*oneoverxj;
   wponxj := wps*oneoverxj;
   wconxj := coeff0+coeff1*wponxj+coeff2*wponxj*wponxj;
   arga := wconxj+djonxj;
   argc := wponxj/(1.0+wponxj);
   argb := sqrt(1.0-argc*argc);
   fshort := 1.0-xjonxl*(arga*argb-djonxj);--Fs
   dwpdvb := model_MOS3coeffDepLayWidth*dsqdvb;
   dadvb := (coeff1+coeff2*(wponxj+wponxj))*dwpdvb*oneoverxj;
   dbdvb := -argc*argc*(1.0-argc)*dwpdvb/(argb*wps);
   dfsdvb := -xjonxl*(dadvb*argb+arga*dbdvb);
ELSE
   fshort := 1.0;
   dfsdvb := 0.0;
END IF;
--*****************************
--*body effect (Lamda_b)
--*****************************
gammas := model_MOS3gamma*fshort;
fbodys := 0.5*gammas/(sqphbs+sqphbs);
fbody := fbodys+model_MOS3narrowFactor/here_MOS3w;
onfbdy := 1.0/(1.0+fbody);
dfbdvb := -fbodys*dsqdvb/sqphbs+fbodys*dfsdvb/fshort;
qbonco := gammas*sqphbs+model_MOS3narrowFactor*phibs/here_MOS3w;
dqbdvb := gammas*dsqdvb+model_MOS3gamma*dfsdvb*sqphbs-
model_MOS3narrowFactor/here_MOS3w;
--************************
--*static feedback effect
--************************
vbix := here_MOS3tVbi*model_MOS3type-eta*(here_MOS3mode*vds);
--*************************
```

```
--*threshold voltage
--*************************
vth := vbix+qbonco;
dvtdvd := -eta;
dvtdvb := dqbdvb;
--************************************************
--*joint weak inversion and strong inversion
--************************************************
von := vth;
IF ( model_MOS3fastSurfaceStateDensity /= 0.0 ) THEN
   csonco := q*model_MOS3fastSurfaceStateDensity*1.0e4*EffectiveLength*here_MOS3w/OxideCap;
   cdonco := qbonco/(phibs+phibs);
   xn := 1.0+csonco+cdonco;
   von := vth+vt*xn;
   dxndvb := dqbdvb/(phibs+phibs)-qbonco*dsqdvb/(phibs*sqphbs);
   dvodvd := dvtdvd;
   dvodvb := dvtdvb+vt*dxndvb;
ELSE
   --**********************
   --*cutoff region
   --**********************
   IF (here_MOS3mode = 1.0) THEN vg := vgs; ELSE vg := vgd; END IF;
   IF ( vg <= von ) THEN
      cdrain := 0.0; here_MOS3gm := 0.0; here_MOS3gds := 0.0;here_MOS3gmbs := 0.0;
   END IF;
END IF;
--*************************
--*device is on
--*************************
IF (here_MOS3mode = 1.0) THEN vg := vgs; ELSE vg := vgd; END IF;
vgsx := max(vg,von);
--*********************************************
--*mobility modulation by gate voltage
--*********************************************
onfg := 1.0+model_MOS3theta*(vgsx-vth);
fgate := 1.0/onfg;
us := here_MOS3tSurfMob * 1.0e-4  *fgate;--/*(m**2/cm**2)*/
dfgdvg := -model_MOS3theta*fgate*fgate;
dfgdvd := -dfgdvg*dvtdvd;
dfgdvb := -dfgdvg*dvtdvb;
--*********************************
--*saturation voltage
--*********************************
vdsat := (vgsx-vth)*onfbdy;
IF ( model_MOS3maxDrIFtVel <= 0.0 ) THEN
   dvsdvg := onfbdy;
   dvsdvd := -dvsdvg*dvtdvd;
   dvsdvb := -dvsdvg*dvtdvb-vdsat*dfbdvb*onfbdy;
ELSE
   vdsc := EffectiveLength*model_MOS3maxDrIFtVel/us;
   onvdsc := 1.0/vdsc;
   arga := (vgsx-vth)*onfbdy;
   argb := sqrt(arga*arga+vdsc*vdsc);
   vdsat := arga+vdsc-argb;
   dvsdga := (1.0-arga/argb)*onfbdy;
   dvsdvg := dvsdga-(1.0-vdsc/argb)*vdsc*dfgdvg*onfg;
   dvsdvd := -dvsdvg*dvtdvd;
   dvsdvb := -dvsdvg*dvtdvb-arga*dvsdga*dfbdvb;
END IF;
--*************************************
```

```
--*current factors in linear region
--************************************
vdsx := min((here_MOS3mode*vds),vdsat);
IF ( vdsx = 0.0 ) THEN
  Beta := Beta*fgate;
  cdrain := 0.0;
  here_MOS3gm := 0.0;
  here_MOS3gds := Beta*(vgsx-vth);
  here_MOS3gmbs := 0.0;
  IF (here_MOS3mode = 1.0) THEN vg := vgs; ELSE vg := vgd; END IF;
  IF ( (model_MOS3fastSurfaceStateDensity /= 0.0) and (vg < von) ) THEN
    here_MOS3gds := here_MOS3gds * exp((vg-von)/(vt*xn));
  END IF;
ELSE
  cdo := vgsx-vth-0.5*(1.0+fbody)*vdsx;
  dcodvb := -dvtdvb-0.5*dfbdvb*vdsx;
  --************************************
  --*normalized drain current
  --************************************
  cdnorm := cdo*vdsx;
  here_MOS3gm := vdsx;
  here_MOS3gds := vgsx-vth-(1.0+fbody+dvtdvd)*vdsx;
  here_MOS3gmbs := dcodvb*vdsx;
  --************************************************************
  --*drain current without velocity saturation effect
  --************************************************************
  cd1 := Beta*cdnorm;
  Beta := Beta*fgate;
  cdrain := Beta*cdnorm;
  here_MOS3gm := Beta*here_MOS3gm+dfgdvg*cd1;
  here_MOS3gds := Beta*here_MOS3gds+dfgdvd*cd1;
  here_MOS3gmbs := Beta*here_MOS3gmbs;
  --************************************
  --*velocity saturation factor
  --************************************
  IF ( model_MOS3maxDrIFtVel /= 0.0 ) THEN
    fdrain := 1.0/(1.0+vdsx*onvdsc);
    fd2 := fdrain*fdrain;
    arga := fd2*vdsx*onvdsc*onfg;
    dfddvg := -dfgdvg*arga;
    dfddvd := -dfgdvd*arga-fd2*onvdsc;
    dfddvb := -dfgdvb*arga;
    --*******************************
    --*drain current
    --*******************************
    here_MOS3gm := fdrain*here_MOS3gm+dfddvg*cdrain;
    here_MOS3gds := fdrain*here_MOS3gds+dfddvd*cdrain;
    here_MOS3gmbs := fdrain*here_MOS3gmbs+dfddvb*cdrain;
    cdrain := fdrain*cdrain;
    Beta := Beta*fdrain;
  END IF;
  --************************************
  --*channel length modulation
  --************************************
  IF (( (here_MOS3mode*vds) <= vdsat ) or (model_MOS3alpha = 0.0) ) THEN --goto line700;
    IF (here_MOS3mode = 1.0) THEN vg := vgs; ELSE vg := vgd; END IF;
    IF ( vg < von ) THEN
      --************************
      --*weak inversion
      --************************
```

```
        onxn := 1.0/xn;
        ondvt := onxn/vt;
        wfact := exp((vg-von)*ondvt );
        cdrain := cdrain*wfact;
        gms := here_MOS3gm*wfact;
        gmw := cdrain*ondvt;
        here_MOS3gm := gmw;
        IF ((here_MOS3mode*vds) > vdsat) THEN
          here_MOS3gm := here_MOS3gm+gds0*dvsdvg*wfact;
        END IF;
        here_MOS3gds := here_MOS3gds*wfact+(gms-gmw)*dvodvd;
        here_MOS3gmbs := here_MOS3gmbs*wfact+(gms-gmw)*dvodvb-gmw*(vg-von)*onxn*dxndvb;
      END IF;
    ELSE
      IF ( model_MOS3maxDrIFtVel <= 0.0 ) THEN --goto line510;
        delxl := sqrt(model_MOS3kappa*((here_MOS3mode*vds)-vdsat)*model_MOS3alpha);
        dldvd := 0.5*delxl/((here_MOS3mode*vds)-vdsat);
        ddldvg := 0.0;
        ddldvd := -dldvd;
        ddldvb := 0.0;
        --****************************************
        --*punch through approximation
        --****************************************
        IF ( delxl > (0.5*EffectiveLength) ) THEN
          delxl := EffectiveLength-(EffectiveLength*EffectiveLength/(4.0*delxl));
          arga := 4.0*(EffectiveLength-delxl)*(EffectiveLength-delxl)/(EffectiveLength*EffectiveLength);
          ddldvg := ddldvg*arga;
          ddldvd := ddldvd*arga;
          ddldvb := ddldvb*arga;
          dldvd :=  dldvd*arga;
        END IF;
        --****************************
        --*saturation region
        --****************************
        dlonxl := delxl*oneoverxl;
        xlfact := 1.0/(1.0-dlonxl);
        cdrain := cdrain*xlfact;
        diddl := cdrain/(EffectiveLength-delxl);
        here_MOS3gm := here_MOS3gm*xlfact+diddl*ddldvg;
        gds0 := here_MOS3gds*xlfact+diddl*ddldvd;
        here_MOS3gmbs := here_MOS3gmbs*xlfact+diddl*ddldvb;
        here_MOS3gm := here_MOS3gm+gds0*dvsdvg;
        here_MOS3gmbs := here_MOS3gmbs+gds0*dvsdvb;
        here_MOS3gds := gds0*dvsdvd+diddl*dldvd;
        --********************************
        --*finish strong inversion case
        --********************************
        IF (here_MOS3mode = 1.0) THEN vg := vgs; ELSE vg := vgd; END IF;
        IF ( vg < von ) THEN
          --*****************************
          --*weak inversion
          --*****************************
          onxn := 1.0/xn;
          ondvt := onxn/vt;
          wfact := exp((vg-von)*ondvt );
          cdrain := cdrain*wfact;
          gms := here_MOS3gm*wfact;
          gmw := cdrain*ondvt;
          here_MOS3gm := gmw;
          IF ((here_MOS3mode*vds) > vdsat) THEN
```

```
        here_MOS3gm := here_MOS3gm+gds0*dvsdvg*wfact;
      END IF;
      here_MOS3gds := here_MOS3gds*wfact+(gms-gmw)*dvodvd;
      here_MOS3gmbs := here_MOS3gmbs*wfact+(gms-gmw)*dvodvb-gmw*(vg-von)*onxn*dxndvb;
    END IF;
  ELSE
    cdsat := cdrain;
    gdsat := cdsat*(1.0-fdrain)*onvdsc;
    gdsat := max(1.0e-12,gdsat);
    gdoncd := gdsat/cdsat;
    gdonfd := gdsat/(1.0-fdrain);
    gdonfg := gdsat*onfg;
    dgdvg := gdoncd*here_MOS3gm-gdonfd*dfddvg+gdonfg*dfgdvg;
    dgdvd := gdoncd*here_MOS3gds-gdonfd*dfddvd+gdonfg*dfgdvd;
    dgdvb := gdoncd*here_MOS3gmbs-gdonfd*dfddvb+gdonfg*dfgdvb;
    --IF (ckt_CKTbadMos3) THEN
    --   emax := cdsat*oneoverxl/gdsat;
    --ELSE
    emax := model_MOS3kappa * cdsat*oneoverxl/gdsat;
    --END IF;
    emoncd := emax/cdsat;
    emongd := emax/gdsat;
    demdvg := emoncd*here_MOS3gm-emongd*dgdvg;
    demdvd := emoncd*here_MOS3gds-emongd*dgdvd;
    demdvb := emoncd*here_MOS3gmbs-emongd*dgdvb;

    arga := 0.5*emax * model_MOS3alpha;
    argc := model_MOS3kappa*model_MOS3alpha;
    argb := sqrt(arga*arga+argc*((here_MOS3mode*vds)-vdsat));
    delxl := argb-arga;
    dldvd := argc/(argb+argb);
    dldem := 0.5*(arga/argb-1.0)*model_MOS3alpha;
    ddldvg := dldem*demdvg;
    ddldvd := dldem*demdvd-dldvd;
    ddldvb := dldem*demdvb;
    IF ( delxl > (0.5*EffectiveLength) ) THEN
      delxl := EffectiveLength-(EffectiveLength*EffectiveLength/(4.0*delxl));
      arga := 4.0*(EffectiveLength-delxl)*(EffectiveLength-delxl)/(EffectiveLength*EffectiveLength);
      ddldvg := ddldvg*arga;
      ddldvd := ddldvd*arga;
      ddldvb := ddldvb*arga;
      dldvd :=  dldvd*arga;
    END IF;
    --**********************
    --*saturation region
    --**********************
    dlonxl := delxl*oneoverxl;
    xlfact := 1.0/(1.0-dlonxl);
    cdrain := cdrain*xlfact;
    diddl := cdrain/(EffectiveLength-delxl);
    here_MOS3gm := here_MOS3gm*xlfact+diddl*ddldvg;
    gds0 := here_MOS3gds*xlfact+diddl*ddldvd;
    here_MOS3gmbs := here_MOS3gmbs*xlfact+diddl*ddldvb;
    here_MOS3gm := here_MOS3gm+gds0*dvsdvg;
    here_MOS3gmbs := here_MOS3gmbs+gds0*dvsdvb;
    here_MOS3gds := gds0*dvsdvd+diddl*dldvd;
    --**********************************
    --*finish strong inversion case
    --**********************************
    IF (here_MOS3mode = 1.0) THEN
```

```
          vg := vgs;
        ELSE
          vg := vgd;
        END IF;
        IF ( vg < von ) THEN
          --************************
          --*weak inversion
          --************************
          onxn := 1.0/xn;
          ondvt := onxn/vt;
          wfact := exp((vg-von)*ondvt );
          cdrain := cdrain*wfact;
          gms := here_MOS3gm*wfact;
          gmw := cdrain*ondvt;
          here_MOS3gm := gmw;
          IF ((here_MOS3mode*vds) > vdsat) THEN
            here_MOS3gm := here_MOS3gm+gds0*dvsdvg*wfact;
          END IF;
          here_MOS3gds := here_MOS3gds*wfact+(gms-gmw)*dvodvd;
          here_MOS3gmbs := here_MOS3gmbs*wfact+(gms-gmw)*dvodvb-gmw*(vg-von)*onxn*dxndvb;
        END IF;
      END IF;
    END IF;
END IF;
--***************************
--*Capacitance Cbs
--***************************
IF(here_MOS3Cbs_C /= 0.0) or (here_MOS3Cbssw_C /= 0.0 ) THEN
  IF (vbs < here_MOS3tDepCap) THEN
     arg:=1.0-vbs/here_MOS3tBulkPot;
     sarg := arg**(-model_MOS3bulkJctBotGradingCoeff);
     sargsw :=arg**(-model_MOS3bulkJctSideGradingCoeff);
     here_MOS3capbs:=here_MOS3Cbs_C * sarg + here_MOS3Cbssw_C*sargsw;
  ELSE
     here_MOS3capbs:=here_MOS3f2s + here_MOS3f3s * vbs;
  END IF;
ELSE
   here_MOS3capbs:= 0.0;
END IF;
MOS3_capbs :=here_MOS3capbs;
--*****************************
--*Capacitance Cbd
--*****************************
IF(here_MOS3Cbd_C /= 0.0) or (here_MOS3Cbdsw_C /= 0.0 ) THEN
  IF (vbd < here_MOS3tDepCap) THEN
     arg:=1.0-vbd/here_MOS3tBulkPot;
     sarg := arg**(-model_MOS3bulkJctBotGradingCoeff);
     sargsw := arg**(-model_MOS3bulkJctSideGradingCoeff);
     here_MOS3capbd:=here_MOS3Cbd_C*sarg+here_MOS3Cbdsw_C*sargsw;
  ELSE
     here_MOS3capbd:=here_MOS3f2d + vbd * here_MOS3f3d;
  END IF;
ELSE
  here_MOS3capbd := 0.0;
END IF;
MOS3_capbd :=here_MOS3capbd;
--************************
--*  meyer's capacitors
--************************
IF (here_MOS3mode > 0.0) THEN
```

```
   here_MOS3capgd:=MEYER_C_CGD(vgs,vgd,von,here_MOS3tPhi,OxideCap,here_MOS3w);
   here_MOS3capgs:=MEYER_C_CGS(vgs,vgd,von,here_MOS3tPhi,OxideCap,here_MOS3w);
   here_MOS3capgb:=MEYER_C_CGB(vgs,vgd,von,here_MOS3tPhi,OxideCap,EffectiveLength);
ELSE
   here_MOS3capgd:=MEYER_C_CGD(vgd,vgs,von,here_MOS3tPhi,OxideCap,here_MOS3w);
   here_MOS3capgs:=MEYER_C_CGS(vgd,vgs,von,here_MOS3tPhi,OxideCap,here_MOS3w);
   here_MOS3capgb:=MEYER_C_CGB(vgd,vgs,von,here_MOS3tPhi,OxideCap,EffectiveLength);
END IF;
MOS3_cgs := here_MOS3capgs;
MOS3_cgd := here_MOS3capgd;
MOS3_cgb := here_MOS3capgb;
--*****************************************************************************************************************
here_MOS3von := model_MOS3type * von;
here_MOS3vdsat := model_MOS3type * vdsat;
here_MOS3cd := model_MOS3type*(here_MOS3mode * cdrain - here_MOS3cbd_i);
MOS3_Id := here_MOS3cd;
MOS3_Is := here_MOS3mode * cdrain - here_MOS3cbs_i;
MOS3_Ib := here_MOS3cbs_i + here_MOS3cbd_i;
MOS3_gm := here_MOS3gm;
MOS3_gds := here_MOS3gds;
MOS3_gmbs := here_MOS3gmbs;
END PROCEDURE;
--*****************************************************************************
FUNCTION Id_MOS3(
             Vdsq, Vgsq, Vbsq :REAL; model_MOS3modName,here_MOS3name:STRING;
             MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs           :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb           :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                 :CAPACITANCE;
BEGIN
EQU_Model_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_Id;
END FUNCTION;
--*****************************************************************************
FUNCTION Is_MOS3(
             Vdsq, Vgsq, Vbsq :REAL; model_MOS3modName,here_MOS3name:STRING;
             MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs           :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb           :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                 :CAPACITANCE;
BEGIN
EQU_Model_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_Is;
END FUNCTION;
--*****************************************************************************
FUNCTION Ib_MOS3(
             Vdsq, Vgsq, Vbsq :REAL;
             model_MOS3modName,here_MOS3name:STRING;  MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs           :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb           :CAPACITANCE;
```

```
VARIABLE MOS3_capbd, MOS3_capbs                    :CAPACITANCE;
BEGIN
EQU_Model_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_Ib;
END FUNCTION;
--*************************************************************************
FUNCTION gds_MOS3(
             Vdsq, Vgsq, Vbsq :REAL;
              model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                   :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs              :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb              :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                    :CAPACITANCE;
BEGIN
EQU_Model_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs );
return MOS3_gds;
END FUNCTION;
--*************************************************************************
FUNCTION gm_MOS3(
             Vdsq, Vgsq, Vbsq :REAL;
             model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                   :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs              :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb              :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                    :CAPACITANCE;
BEGIN
EQU_Model_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_gm;
END FUNCTION;
--*************************************************************************
FUNCTION gmbs_MOS3(
             Vdsq, Vgsq, Vbsq :REAL;
             model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
            ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                   :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs              :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb              :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                    :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
            Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_gmbs;
END FUNCTION;
--*************************************************************************
FUNCTION MOS3_C_CGD(
              Vdsq, Vgsq, Vbsq :REAL;
              model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
             ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                   :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs              :REAL;
```

```
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb          :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
        Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
        MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
        MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_cgd;
END FUNCTION;
--*************************************************************************
FUNCTION MOS3_C_CGS(
            Vdsq, Vgsq, Vbsq :REAL;
            model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
           ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib               :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs          :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb          :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
        Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
        MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
        MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs );
return MOS3_cgs;
END FUNCTION;
--*************************************************************************
FUNCTION MOS3_C_CGB(
            Vdsq, Vgsq, Vbsq :REAL;
            model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
           ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib               :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs          :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb          :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
        Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
        MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
        MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_cgb;
END FUNCTION;
--*************************************************************************
FUNCTION MOS3_C_CBD(
            Vdsq, Vgsq, Vbsq :REAL;
            model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
           ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib               :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs          :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb          :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
        Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
        MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
        MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_capbd;
END FUNCTION;
--*************************************************************************
FUNCTION MOS3_C_CBS(
            Vdsq, Vgsq, Vbsq :REAL;
```

```
                model_MOS3modName,here_MOS3name:STRING; MOS3_TEMP :REAL
               ) RETURN REAL IS
VARIABLE MOS3_Id,MOS3_Is,MOS3_Ib                :REAL;
VARIABLE MOS3_gm, MOS3_gds, MOS3_gmbs           :REAL;
VARIABLE MOS3_cgd, MOS3_cgs, MOS3_cgb           :CAPACITANCE;
VARIABLE MOS3_capbd, MOS3_capbs                 :CAPACITANCE;
BEGIN
EQU_Model_MOS3(
            Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP,
            MOS3_Id,MOS3_Is,MOS3_Ib,MOS3_gm, MOS3_gds, MOS3_gmbs,
            MOS3_cgd, MOS3_cgs, MOS3_cgb, MOS3_capbd, MOS3_capbs);
return MOS3_capbs;
END FUNCTION;
--*****************************************************************************
FUNCTION R_RD_T(defaultNRD,defaultM,R_TEMP:REAL ) return REAL IS
CONSTANT NRD :REAL:=defaultNRD;
CONSTANT M   :REAL:=defaultM;
VARIABLE delta_T, RDeff, arg, RD_T:REAL;
--*****************************
--*  Device parameters
--*****************************
VARIABLE here_MOS3drainSquares :REAL:= NRD;
VARIABLE here_DraiContRes :REAL:= RDC;
VARIABLE here_NumParaTra  :REAL:= M;
--*************************
--*  Model parameters
--*************************
VARIABLE model_Rtnom :REAL:= TNOM;
VARIABLE model_RtempCoeffLinearRD     :REAL:= TRD1;
VARIABLE model_RtempCoeffQuadraticRD  :REAL:= TRD2;
VARIABLE model_MOS3sheetResistance    :REAL:= RSH;
VARIABLE model_MOS3drainResistance    :REAL:= RD;
BEGIN
 IF(model_Rtnom = UNDEF) THEN
  model_Rtnom := defaulttnom + CONSTCtoK;
 ELSE
  model_Rtnom := model_Rtnom + CONSTCtoK;
 END IF;
 delta_T := R_TEMP - model_Rtnom;
 arg := 1.0 + model_RtempCoeffLinearRD  * delta_T + model_RtempCoeffQuadraticRD  * delta_T**2;
 IF(model_MOS3drainResistance /= UNDEF) THEN
  IF(model_MOS3drainResistance /= 0.0) THEN
   model_MOS3drainResistance := RD;
  ELSE
   model_MOS3drainResistance := 0.0;
  END IF;
 ELSIF (model_MOS3sheetResistance /= UNDEF) THEN
  IF(model_MOS3sheetResistance /= 0.0) THEN
    IF (here_MOS3drainSquares > 0.0) THEN
      model_MOS3drainResistance := model_MOS3sheetResistance*here_MOS3drainSquares;
    ELSE
     REPORT "ERROR FATAL: Squares of drain resistance <= 0.0 !!!"
     SEVERITY ERROR; --WARNING;
    END IF;
  ELSE
    model_MOS3drainResistance := 0.0;
  END IF;
 ELSE
  model_MOS3drainResistance:= 0.0;
 END IF;
```

```
  RDeff := ( model_MOS3drainResistance + here_DraiContRes) / here_NumParaTra;
  RD_T := RDeff * arg;
  return RD_T;
END FUNCTION;
--*****************************************************************************
FUNCTION R_RS_T(defaultNRS,defaultM,R_TEMP :REAL) return REAL IS
CONSTANT NRS :REAL:=defaultNRS;
CONSTANT M     :REAL:=defaultM;
VARIABLE delta_T, RSeff, arg, RS_T:REAL;
--**************************
--*  Device parameters
--**************************
VARIABLE here_MOS3sourceSquares :REAL:= NRS;
VARIABLE here_SourContRes :REAL:= RSC;
VARIABLE here_NumParaTra  :REAL:=M;
--**********************
--*  Model parameters *
--**********************
VARIABLE model_Rtnom :REAL:= TNOM;
VARIABLE model_RtempCoeffLinearRS     :REAL:=TRS1;
VARIABLE model_RtempCoeffQuadraticRS  :REAL:=TRS2;
VARIABLE model_MOS3sheetResistance    :REAL:= RSH;
VARIABLE model_MOS3sourceResistance   :REAL:= RS;
BEGIN
  IF(model_Rtnom = UNDEF) THEN
    model_Rtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Rtnom := model_Rtnom + CONSTCtoK;
  END IF;
  delta_T := R_TEMP - model_Rtnom;
  arg := 1.0 + model_RtempCoeffLinearRS  * delta_T + model_RtempCoeffQuadraticRS  * delta_T**2;
  IF(model_MOS3sourceResistance /= UNDEF ) THEN
     IF(model_MOS3sourceResistance /= 0.0) THEN
       model_MOS3sourceResistance := RS;
     ELSE
       model_MOS3sourceResistance := 0.0;
     END IF;
  ELSIF (model_MOS3sheetResistance /= UNDEF) THEN
     IF(model_MOS3sheetResistance /= 0.0) THEN
        IF (here_MOS3sourceSquares > 0.0) THEN
          model_MOS3sourceResistance := model_MOS3sheetResistance*here_MOS3sourceSquares;
        ELSE
          REPORT "ERROR FATAL: Squares of source resistance <= 0.0 !!!"
          SEVERITY ERROR; --WARNING;
        END IF;
     ELSE
        model_MOS3sourceResistance:= 0.0;
     END IF;
  ELSE
    model_MOS3sourceResistance := 0.0;
  END IF;
  RSeff := (model_MOS3sourceResistance + here_SourContRes) / here_NumParaTra;
  RS_T := RSeff * arg;
  return RS_T;
END FUNCTION;
--*****************************************************************************
FUNCTION INFORMATION (OK:REAL; modelName,Name:STRING) return REAL IS
BEGIN
IF (OK =1.0) THEN
  REPORT modelName SEVERITY NOTE;
```

```
  REPORT Name SEVERITY NOTE;
  return OK;
ELSE
  return OK;
END IF;
END FUNCTION;
--*****************************************************************************
terminal  d1, s1: electrical;
CONSTANT model_MOS3modName :STRING:="NMOD";
CONSTANT here_MOS3name :STRING:="NMOD";
QUANTITY OK :REAL;
--*************************
--* FREE QUANTITY
--*************************
  QUANTITY  gm, gds, gmbs                :REAL;
  QUANTITY R_RD, R_RS                    :RESISTANCE;
  QUANTITY C_CGD, C_CGS, C_CGB           :CAPACITANCE;
  QUANTITY C_CBD, C_CBS                  :CAPACITANCE;
--***********************************
--* ELECTRICAL QUANTITY
--***********************************
  QUANTITY Vdsq ACROSS D1 TO S1;
  QUANTITY Vgsq ACROSS gate TO S1;
  QUANTITY Vbsq ACROSS bulk TO S1;
  QUANTITY Idq THROUGH D1;
  QUANTITY Isq THROUGH S1;
  QUANTITY Ibq THROUGH bulk;
  --Capacitance
  QUANTITY Vcgd ACROSS Icgd THROUGH gate TO D1;
  QUANTITY Vcgs ACROSS Icgs THROUGH gate TO S1;
  QUANTITY Vcgb ACROSS Icgb THROUGH gate TO bulk;
  QUANTITY Vcbd ACROSS Icbd THROUGH bulk TO D1;
  QUANTITY Vcbs ACROSS Icbs THROUGH bulk TO S1;
  -- Resistance
  QUANTITY Vrs ACROSS Irs THROUGH S1 TO source;
  QUANTITY Vrd ACROSS Ird THROUGH drain TO D1;

  QUANTITY DomainType: REAL:=0.0;
--*********************************
--* THERMAL QUANTITY
--*********************************
  --QUANTITY MOS3_TEMP ACROSS MOS3_P THROUGH T1 TO THERMAL_REF;
  CONSTANT MOS3_TEMP :TEMPERATURE:=defaulttemp;
  CONSTANT TEMP1 :TEMPERATURE:= MOS3_TEMP + CONSTCtoK;
BEGIN
 -- Current Ids
 Idq  == Id_MOS3(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP +
           CONSTCtoK );--+ defaulttemp
 Isq  == Is_MOS3(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP +
          CONSTCtoK );
 Ibq  == Ib_MOS3(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP +
          CONSTCtoK );
 gds  == gds_MOS3(Vdsq, Vgsq, Vbsq ,model_MOS3modName,here_MOS3name,MOS3_TEMP +
           CONSTCtoK);
 gm   == gm_MOS3(Vdsq, Vgsq, Vbsq,model_MOS3modName,here_MOS3name,MOS3_TEMP +
            CONSTCtoK );
 gmbs == gmbs_MOS3(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP +
             CONSTCtoK);

IF (DOMAIN = TIME_DOMAIN) or (DOMAIN = FREQUENCY_DOMAIN) USE
```

```
   -- TIME DOMAIN and FREQUENCY DOMAIN
   DomainType == 1.0;
   -- Capacitance
   C_CGD == MOS3_C_CGD(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,TEMP1 );
   C_CGS == MOS3_C_CGS(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,TEMP1 );
   C_CGB == MOS3_C_CGB(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,TEMP1 );
   C_CBD == MOS3_C_CBD(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP
+ CONSTCtoK);
   C_CBS == MOS3_C_CBS(Vdsq, Vgsq, Vbsq, model_MOS3modName,here_MOS3name,MOS3_TEMP
+ CONSTCtoK);

   Icgd == C_CGD * Vcgd'dot;
   Icgs == C_CGS * Vcgs'dot;
   Icgb == C_CGB * Vcgb'dot;
   Icbd == C_CBD * Vcbd'dot;
   Icbs == C_CBS * Vcbs'dot;
 ELSE
   -- QUIESCENT DOMAIN
   DomainType == 2.0;
   -- Capacitance
   C_CGD == 0.0;
   C_CGS == 0.0;
   C_CGB == 0.0;
   C_CBD == 0.0;
   C_CBS == 0.0;
   Icgd == 0.0;
   Icgs == 0.0;
   Icgb == 0.0;
   Icbd == 0.0;
  Icbs == 0.0;
 END USE;
 -- Resistance
 R_RS == R_RS_T(defaultNRS,defaultM,MOS3_TEMP + CONSTCtoK );
 R_RD == R_RD_T(defaultNRD,defaultM,MOS3_TEMP + CONSTCtoK );
 Vrs == Irs * r_rs;
 Vrd == Ird * r_rd;

  -- Calcul de la puissance dissipee
 -- MOS3_P == abs((Vrs + Vrd + Vdsq) * Idq);
 --information sur le circuit
 OK == INFORMATION (1.0,model_MOS3modName,here_MOs3name);
END mos3;
```

Annexe 2

MODELE VHDL-AMS D'UNE CELLULE MEMOIRE DE TYPE EEPROM

```
LIBRARY IEEE; USE IEEE.math_REAL.all;
LIBRARY STD; USE STD.standard.all;
USE work.electrical_systems.all;
USE work.thermal_systems.all;
USE work.CellData.all;
ENTITY CELL_L1 IS
  GENERIC(
        L               :REAL:= defaultCellL;       --Longueur de canal (m)
        W               :REAL:= defaultCellW;       --Largeur de canal (m)
        AS              :REAL:= 0.0;                --Surface du source (m2)
        AD              :REAL:= 0.0;                --Surface du drain (m2)
        LD              :REAL:= 0.0;                --Diffusion latérale (m)
        CGDO            :CAPACITANCE:= 0.0;         --Capacité de débordement G-D (F)
        CGSO            :CAPACITANCE:= 0.0;         --Capacité de débordement G-S (F)
        CGBO            :CAPACITANCE:= 0.0;         --Capacité de débordement G-B (F)
        VJ              :VOLTAGE:= 1.0;             -- Barrière de potentiel (Volt)
        MJ              :REAL:= 0.0;                -- Facteur de gradualité (-)
        Stun            :REAL:= 0.0;                --Surface de la zone tunnel (m2)
        Ttun            :REAL:= 1.0e-7;             --Epaisseur de la couche d'oxyde tunnel (m)
        Spp             :REAL:= 0.0;                --Surface zone formée par la GF et la GC (m2)
        Tpp             :REAL:= 1.0e-7;             --Epaisseur couche oxyde entre GC et GF (m)
        GAMMA           :REAL:= 0.526;    --Effet de la polarisation de substrat sur la tension de seuil (Volt1/2)
        VTO             :VOLTAGE := 1.0;            --Tension de seuil à polarisation de substrat nulle (Volt)
        PHI             :VOLTAGE:= 0.58;    --Potentiel de la surface dans la région de forte inversion (Volt)
        KP              :REAL:= 0.0;                --La transconductance (A/Volt2)
        LAMBDA          :REAL:= 0.0;                --Modulation de la langueur du canal (Volt-1)
        RD              :RESISTANCE :=0.0;          --Résistance du drain (Ohm)
        RS              :RESISTANCE :=0.0;          --Résistance de la source (Ohm)
        RSH             :RESISTANCE :=0.0;          --Résistance/carrée des diffusions de S et D (Ohm/□)
        TRD1            :REAL:=0.0;                 --Coeff. de température (linaire) RD (1/°K)
        TRD2            :REAL:=0.0;                 --Coeff. de température (quadratique) RD (1/°K2)
        TRS1            :REAL:=0.0;                 --Coeff. de température (linaire) RS (1/°K)
        TRS2            :REAL:=0.0;                 --Coeff. de température (quadratique) RS  (1/°K2)
        RDC             :RESISTANCE :=0.0;          --Résistance du drain due au résistivité du contact (Ohm)
        RSC             :RESISTANCE :=0.0 ;   --Résistance de la source due au résistivité du contact (Ohm)
        IS_o            :CURRENT:=1.0E-14;          --Courant de saturation des diodes (A)
        JS              :REAL:=0.0;                 --Densité de courant de saturation des diodes (A/m)
        U0              :REAL:=600.0;               --Mobilité des porteurs (cm2/(V.s))
        TOX             :REAL:=1.0E-7;              --Epaisseur de l'oxyde sous la grille (m)
        BEX             :REAL:=-1.5;                --Coeff. de température pour mobilité
```

```
        ALPHAECR   :REAL:=0.0;              --Paramètre multiplicatif tunnel en écriture (A/V2)
        BETAECR    :REAL:=0.0;              --Paramètre exponentiel tunnel en écriture (V/m)
        ALPHAEFF   :REAL:=0.0;              --Paramètre multiplicatif tunnel en effacement (A/V2)
        BETAEFF    :REAL:=0.0;              --Paramètre exponentiel tunnel en effacement (V/m)
        THETA      :REAL:=0.0;              --Modulation de la mobilité (1/Volt)
        NFS        :REAL:=0.0;              --Densité d'état de surface rapide (AT/cm2)
        QFG0       :REAL:=0.0;              --Charge initiale de la grille flottante (C)
        NSUB       :REAL:=UNDEF;            --Dopage du substrat (AT/cm3)
        TNOM       :REAL:=UNDEF             --Température nominale (°C)
         );
  PORT (Terminal Drain, gc, Source, Bulk :electrical
          --Terminal T2,T1:thermal
       );
END CELL_L1;
ARCHITECTURE Arch_de_Cell OF CELL_L1 IS
--*****************************************************************************
FUNCTION max(a,b:REAL) return REAL IS
BEGIN
 IF (a>b) THEN return a;
 ELSE return b;
 END IF;
END FUNCTION;
--*****************************************************************************
FUNCTION min(a,b:REAL) return REAL IS
BEGIN
 IF (a>b) THEN return b;
 ELSE return a;
 END IF;
END FUNCTION;
--*****************************************************************************
FUNCTION MEYER_C_CGB(Vfgs,Vds,Von:VOLTAGE;
                phi_1,cox_1:REAL;
                Leff:REAL
               ) return CAPACITANCE IS

VARIABLE Cgb, GateBulkOverlapCap :CAPACITANCE;
VARIABLE model_CellgateBulkOverlapCapFactor :REAL:= CGBO;
BEGIN
 GateBulkOverlapCap := model_CellgateBulkOverlapCapFactor * Leff;
 --Calcule de la capacite de CGB
 IF (Vfgs < (Von - phi_1)) THEN                          -- Accumulation region
   Cgb := cox_1 + GateBulkOverlapCap;
 ELSIF (Vfgs > (Von - phi_1)) and (Vfgs < Von) THEN      -- Depletion region
   Cgb := cox_1 *((Von - Vfgs)/phi_1) + GateBulkOverlapCap;
 ELSIF (Vfgs > Von) and (Vfgs < (Von + Vds)) THEN        -- Saturation region
   Cgb := GateBulkOverlapCap;
 ELSE                                                    -- Linear region
   Cgb := GateBulkOverlapCap;
 END IF;
return Cgb;
END FUNCTION;
--*****************************************************************************
FUNCTION MEYER_C_CGS(Vfgs,Vds,Von:VOLTAGE;
                phi_1, cox_1:REAL;
                Weff:REAL
               ) return CAPACITANCE IS
VARIABLE Cgs, GateSourceOverlapCap :CAPACITANCE;
VARIABLE model_CellgateSourceOverlapCapFactor :REAL:= CGSO;
BEGIN
 GateSourceOverlapCap := model_CellgateSourceOverlapCapFactor * Weff;
```

```
   --Calcule de la capacite de CGS
  IF (Vfgs < (Von - phi_1) )  THEN                                  -- Accumulation region
     Cgs := GateSourceOverlapCap;
  ELSIF (Vfgs > (Von - phi_1)) and (Vfgs < Von)  THEN               -- Depletion region
     Cgs := 2.0/3.0*cox_1*(1.0-(Von - Vfgs)/phi_1) + GateSourceOverlapCap;
  ELSIF (Vfgs > Von) and (Vfgs < (Von + Vds)) THEN                  -- Saturation region
     Cgs := 2.0/3.0*cox_1 + GateSourceOverlapCap ;
  ELSE                                                              -- Linear region
     Cgs := 2.0/3.0*cox_1*(1.0 - ( (Vfgs - Vds - Von)/(2.0*(Vfgs - Von) - Vds) )**2) +
GateSourceOverlapCap;
  END IF;
return Cgs;
END FUNCTION;

--*************************************************************************
FUNCTION MEYER_C_CGD(Vfgs,Vds,Von:VOLTAGE;
                                  phi_1, cox_1:REAL;
                                  Weff:REAL) return CAPACITANCE IS

VARIABLE GateDrainOverlapCap, Cgd :CAPACITANCE;
VARIABLE model_CellgateDrainOverlapCapFactor :REAL:= CGDO;
BEGIN
  GateDrainOverlapCap := model_CellgateDrainOverlapCapFactor * Weff;
  -- Calcule de la capacite de CGD
  IF (Vfgs < (Von - phi_1)) THEN                                    --Accumulation region
     Cgd := GateDrainOverlapCap;
  ELSIF (Vfgs > (Von - phi_1)) and (Vfgs < Von) THEN                --Depletion region
     Cgd := GateDrainOverlapCap;
  ELSIF (Vfgs > Von) and (Vfgs < (Von + Vds)) THEN                  --Saturation region
     Cgd := GateDrainOverlapCap;
  ELSE                                                              --Linear region
     Cgd := 2.0/3.0*cox_1*(1.0 - ( ((Vfgs - Von))/(2.0*(Vfgs - Von) - Vds) )**2) + GateDrainOverlapCap;
  END IF;
return Cgd;
END FUNCTION;
--*************************************************************************
FUNCTION TUNNEL_C_CTUN(Vfgs,Vds:VOLTAGE) return CAPACITANCE IS

VARIABLE Ctunmax, Ctun:CAPACITANCE;
VARIABLE model_CellGradingCoeff       :REAL:= MJ;
VARIABLE model_CellJctPotential       :REAL:= VJ;
VARIABLE model_CellTunelArea          :REAL:=Stun;
VARIABLE model_CellTunelOxideThickness:REAL:=Ttun;
BEGIN
 Ctunmax := EPSOX * model_CellTunelArea /model_CellTunelOxideThickness;
 -- Calcule de la capacite de Ctun
  IF ((Vfgs - Vds) >= 0.0) THEN                                      --Accumulation region
     Ctun := Ctunmax;
  ELSIF (((Vfgs - Vds) > -5.0) and ((Vfgs - Vds) < 0.0)) THEN        --Depletion region
     Ctun := Ctunmax /(1.0 + (Vds - Vfgs)/model_CellJctPotential)**model_CellGradingCoeff;
  ELSIF ((Vfgs - Vds) <= -5.0) THEN
     Ctun := Ctunmax /(1.0 + (5.0)/model_CellJctPotential)**model_CellGradingCoeff;
  END IF;
  return Ctun;
END FUNCTION;
--*************************************************************************
FUNCTION INTERPOLY_C_CPP(Vgc,Vfg:VOLTAGE
                    ) return CAPACITANCE IS

VARIABLE Cppmax, Cpp:CAPACITANCE;
```

```
VARIABLE model_CellGradingCoeff       :REAL:= MJ;
VARIABLE model_CellJctPotential       :REAL:= VJ;
VARIABLE model_CellInterPolyArea          :REAL:=Spp;
VARIABLE model_CellInterPolyOxideThickness:REAL:=Tpp;
BEGIN
  Cppmax := EPSOX * model_CellInterPolyArea /model_CellInterPolyOxideThickness;
  -- Calcule de la capacite de Cpp
  IF ((Vfg - Vgc) >= 0.0) THEN                    -- Accumulation region
     Cpp := Cppmax;
  ELSIF ((Vfg-Vgc) >(-5.0) and ((Vfg-Vgc) <(0.0))) THEN                  -- Depletion region
     Cpp := Cppmax /(1.0 + (Vgc - Vfg)/model_CellJctPotential)**model_CellGradingCoeff;
  ELSIF ((Vfg - Vgc) <= -5.0) THEN
     Cpp := Cppmax /(1.0 + (5.0)/model_CellJctPotential)**model_CellGradingCoeff;
  END IF;
  return Cpp;
END FUNCTION;
--*****************************************************************************
FUNCTION R_RD_T(R_TEMP:REAL) return REAL IS

constant NRD :REAL:=defaultNRD;
constant M   :REAL:=defaultM;
VARIABLE delta_T, RDeff, arg, RD_T:REAL;
--***************************
--*  Device parameters  *
--***************************
VARIABLE here_CelldrainSquares :REAL:= NRD;
VARIABLE here_DraiContRes :REAL:= RDC;
VARIABLE here_NumParaTra  :REAL:= M;
--*************************
--*  Model parameters *
--*************************
VARIABLE model_Rtnom :REAL:= TNOM;
VARIABLE model_RtempCoeffLinearRD     :REAL:= TRD1;
VARIABLE model_RtempCoeffQuadraticRD  :REAL:= TRD2;
VARIABLE model_CellsheetResistance    :REAL:= RSH;
VARIABLE model_CelldrainResistance    :REAL:= RD;
BEGIN
  IF(model_Rtnom = UNDEF) THEN
    model_Rtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Rtnom := model_Rtnom + CONSTCtoK;
  END IF;

  delta_T := R_TEMP - model_Rtnom;
  arg := 1.0 + model_RtempCoeffLinearRD  * delta_T + model_RtempCoeffQuadraticRD  * delta_T**2;

  IF(model_CelldrainResistance /= UNDEF) THEN
    IF(model_CelldrainResistance /= 0.0) THEN
      model_CelldrainResistance := RD;
    ELSE
      model_CelldrainResistance := 0.0;
    END IF;
  ELSIF (model_CellsheetResistance /= UNDEF) THEN
    IF(model_CellsheetResistance /= 0.0) THEN
      IF (here_CelldrainSquares > 0.0) THEN
        model_CelldrainResistance := model_CellsheetResistance*here_CelldrainSquares;
      ELSE
        REPORT "ERROR FATAL: Squares of drain resistance <= 0.0 !!!"
        SEVERITY ERROR;
      END IF;
```

```
    ELSE
      model_CelldrainResistance := 0.0;
    END IF;
  ELSE
    model_CelldrainResistance:= 0.0;
  END IF;

  RDeff := ( model_CelldrainResistance + here_DraiContRes) / here_NumParaTra;

  RD_T := RDeff * arg;
  return RD_T;
END FUNCTION;
--*****************************************************************************
FUNCTION R_RS_T(R_TEMP :TEMPERATURE) return REAL IS

constant NRS :REAL:=defaultNRS;
constant M   :REAL:=defaultM;
VARIABLE delta_T, RSeff, arg, RS_T:REAL;
--*************************
--*  Device parameters *
--*************************
VARIABLE here_CellsourceSquares :REAL:= NRS;
VARIABLE here_SourContRes :REAL:= RSC;
VARIABLE here_NumParaTra  :REAL:=M;
--*************************
--*  Model parameters *
--*************************
VARIABLE model_Rtnom :TEMPERATURE:= TNOM;
VARIABLE model_RtempCoeffLinearRS     :REAL:=TRS1;
VARIABLE model_RtempCoeffQuadraticRS  :REAL:=TRS2;
VARIABLE model_CellsheetResistance    :REAL:= RSH;
VARIABLE model_CellsourceResistance   :REAL:= RS;
BEGIN
  IF(model_Rtnom = UNDEF) THEN
    model_Rtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Rtnom := model_Rtnom + CONSTCtoK;
  END IF;

  delta_T := R_TEMP - model_Rtnom;
  arg := 1.0 + model_RtempCoeffLinearRS  * delta_T + model_RtempCoeffQuadraticRS  * delta_T**2;
  IF(model_CellsourceResistance /= UNDEF ) THEN
     IF(model_CellsourceResistance /= 0.0) THEN
       model_CellsourceResistance := RS;
     ELSE
       model_CellsourceResistance := 0.0;
     END IF;
  ELSIF (model_CellsheetResistance /= UNDEF) THEN
     IF(model_CellsheetResistance /= 0.0) THEN
        IF (here_CellsourceSquares > 0.0) THEN
          model_CellsourceResistance := model_CellsheetResistance*here_CellsourceSquares;
        ELSE
          REPORT "ERROR FATAL: Squares of source resistance <= 0.0 !!!"
          SEVERITY ERROR;
        END IF;
     ELSE
        model_CellsourceResistance:= 0.0;
     END IF;
  ELSE
    model_CellsourceResistance := 0.0;
```

```
  END IF;
  RSeff := (model_CellsourceResistance + here_SourContRes) / here_NumParaTra;
  RS_T := RSeff * arg;
  return RS_T;
END FUNCTION;
--*****************************************************************************
FUNCTION IBS_DIODE (Vbs :VOLTAGE; D_TEMP:TEMPERATURE
                                )RETURN CURRENT IS

VARIABLE model_Celltnom :REAL:=TNOM;
VARIABLE model_CelljctSatCur     :CURRENT:=IS_o;
VARIABLE model_CelljctSatCurDensity:REAL:=JS;

VARIABLE here_Cellgbs   :REAL;
VARIABLE here_Cellcbs_i :CURRENT;
VARIABLE here_Celltemp  :REAL:=D_TEMP;
VARIABLE here_CellsourceArea :REAL:=AS;

VARIABLE here_CelltSatCurDens :REAL;
VARIABLE here_CelltSatCur:REAL;

VARIABLE SourceSatCur,evbs,vtnom,vt,egfet1,egfet :REAL;
BEGIN
  IF(model_Celltnom = UNDEF) THEN
    model_Celltnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Celltnom := model_Celltnom + CONSTCtoK;
  END IF;
  IF(here_Celltemp = UNDEF) THEN here_Celltemp := defaulttemp + CONSTCtoK; END IF;

  vtnom := model_Celltnom * CONSTKoverQ;
  vt := here_Celltemp * CONSTKoverQ;
  egfet1 := 1.16-(7.02e-4*model_Celltnom*model_Celltnom)/(model_Celltnom+1108.0);
  egfet := 1.16-(7.02e-4*here_Celltemp*here_Celltemp)/(here_Celltemp+1108.0);
  here_CelltSatCur := model_CelljctSatCur * exp(-egfet/vt+egfet1/vtnom);
  here_CelltSatCurDens := model_CelljctSatCurDensity * exp(-egfet/vt+egfet1/vtnom);

  IF( (here_CelltSatCurDens = 0.0) or (here_CellsourceArea = 0.0)) THEN
    SourceSatCur := here_CelltSatCur;
  ELSE
    SourceSatCur := here_CelltSatCurDens * here_CellsourceArea;
  END IF;

  IF(Vbs <= 0.0) THEN
    here_Cellgbs := SourceSatCur/vt;
    here_Cellcbs_i := here_Cellgbs*Vbs;  --Bulk-Source Current
    here_Cellgbs :=  here_Cellgbs + ckt_CKTgmin;
  ELSE
    evbs := exp(MIN(MAX_EXP_ARG,Vbs/vt));
    here_Cellgbs := SourceSatCur*evbs/vt + ckt_CKTgmin;
    here_Cellcbs_i := SourceSatCur * (evbs-1.0);
  END IF;
  return here_Cellcbs_i;
END FUNCTION;
--*****************************************************************************
FUNCTION IBD_DIODE (Vbd :VOLTAGE; D_TEMP:TEMPERATURE
                                )RETURN CURRENT IS

VARIABLE model_Celltnom :REAL:=TNOM;
VARIABLE model_CelljctSatCur     :CURRENT:=IS_o;
```

```
VARIABLE model_CelljctSatCurDensity:REAL:=JS;

VARIABLE here_Cellgbd   :REAL;
VARIABLE here_Cellcbd_i :CURRENT;
VARIABLE here_Celltemp  :REAL:=D_TEMP;
VARIABLE here_CelldrainArea :REAL:=AD;

VARIABLE here_CelltSatCurDens :REAL;
VARIABLE here_CelltSatCur:REAL;

VARIABLE evbd,DrainSatCur,vtnom,vt,egfet1,egfet :REAL;
BEGIN
  IF(model_Celltnom = UNDEF) THEN
    model_Celltnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Celltnom := model_Celltnom + CONSTCtoK;
  END IF;
IF(here_Celltemp = UNDEF) THEN here_Celltemp := defaulttemp + CONSTCtoK; END IF;

  vtnom := model_Celltnom * CONSTKoverQ;
  vt := here_Celltemp * CONSTKoverQ;
  egfet1 := 1.16-(7.02e-4*model_Celltnom*model_Celltnom)/(model_Celltnom+1108.0);
  egfet := 1.16-(7.02e-4*here_Celltemp*here_Celltemp)/(here_Celltemp+1108.0);
  here_CelltSatCur := model_CelljctSatCur * exp(-egfet/vt+egfet1/vtnom);
  here_CelltSatCurDens := model_CelljctSatCurDensity * exp(-egfet/vt+egfet1/vtnom);

  IF( (here_CelltSatCurDens = 0.0) or (here_CelldrainArea = 0.0)) THEN
    DrainSatCur := here_CelltSatCur;
  ELSE
    DrainSatCur := here_CelltSatCurDens * here_CelldrainArea;
  END IF;

  IF(Vbd <= 0.0) THEN
    here_Cellgbd := DrainSatCur/vt;
    here_Cellcbd_i := here_Cellgbd*Vbd;  --Bulk-Drain Current
    here_Cellgbd :=  here_Cellgbd + ckt_CKTgmin;
  ELSE
    evbd := exp(MIN(MAX_EXP_ARG,Vbd/vt));
    here_Cellgbd := DrainSatCur   * evbd/vt + ckt_CKTgmin;
    here_Cellcbd_i := DrainSatCur * (evbd-1.0);
  END IF;
  return  here_Cellcbd_i;
END FUNCTION;
--*****************************************************************************
FUNCTION PHI_FUN(PHI_TEMP :TEMPERATURE) RETURN REAL IS

VARIABLE model_Cellphi            :VOLTAGE:=PHI;
VARIABLE model_CellsubstrateDoping :REAL:=NSUB;
VARIABLE model_Celltnom           :TEMPERATURE:=TNOM;
VARIABLE here_Celltemp            :TEMPERATURE:=PHI_TEMP;
VARIABLE here_CelltPhi            :REAL;
VARIABLE vt,ratio, fact2, kt, egfet, arg, pbfact, vtnom, phio :REAL;
VARIABLE fact1, kt1, egfet1, arg1, pbfact1 :REAL;
BEGIN
  IF(model_Celltnom = UNDEF) THEN
    model_Celltnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Celltnom := model_Celltnom + CONSTCtoK;
  END IF;
  vtnom := model_Celltnom*CONSTKoverQ;
```

```
  IF( model_CellsubstrateDoping/= UNDEF) THEN
    IF(model_CellsubstrateDoping*1.0e6 > 1.45e16) THEN --/*(cm**3/m**3)*/
      IF(model_Cellphi = UNDEF) THEN
        model_Cellphi := 2.0*vtnom*log(model_CellsubstrateDoping*1.0e6/1.45e16);--(cm**3/m**3)
        model_Cellphi := max(0.1,model_Cellphi);
      END IF;
    END IF;
  ELSE
     model_CellsubstrateDoping := 0.0;
     report "ERROR FATAL : Nsub < Ni or not defined"
       severity ERROR; --WARNING;
  END IF;

  fact1 := model_Celltnom/REFTEMP;
  kt1 := CONSTboltz * model_Celltnom;
  egfet1 := 1.16-(7.02e-4*model_Celltnom*model_Celltnom)/(model_Celltnom+1108.0);
  arg1 := -egfet1/(kt1+kt1)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
  pbfact1 := -2.0*vtnom *(1.5*log(fact1)+q*arg1);

  vt := here_Celltemp * CONSTKoverQ;
  ratio := here_Celltemp/model_Celltnom;
  fact2 := here_Celltemp/REFTEMP;
  kt := here_Celltemp * CONSTboltz;
  egfet := 1.16-(7.02e-4*here_Celltemp*here_Celltemp)/(here_Celltemp+1108.0);
  arg := -egfet/(kt+kt)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
  pbfact := -2.0*vt *(1.5*log(fact2)+q*arg);
  phio := (model_Cellphi-pbfact1)/fact1;
  here_CelltPhi := fact2 * phio + pbfact;
  RETURN here_CelltPhi;
END FUNCTION;
--*****************************************************************************************************
FUNCTION GAMMA_FUN RETURN REAL IS

VARIABLE model_Cellgamma          :REAL:=GAMMA;
VARIABLE model_CellsubstrateDoping :REAL:=NSUB;
VARIABLE model_CelloxideThickness     :REAL:=TOX;
VARIABLE model_CelloxideCapFactor     :REAL;
BEGIN
 IF( model_CellsubstrateDoping/= UNDEF) THEN
   IF(model_CellsubstrateDoping*1.0e6 > 1.45e16) THEN --/*(cm**3/m**3)*/
     IF(model_Cellgamma = UNDEF) THEN
       model_CelloxideCapFactor := 3.9 * 8.854214871e-12/model_CelloxideThickness;
       model_Cellgamma := sqrt(2.0 * EPSSIL * q * model_CellsubstrateDoping*1.0e6)/
model_CelloxideCapFactor;--/*(cm**3/m**3)*/
     END IF;
   END IF;
 ELSE
    model_CellsubstrateDoping := 0.0;
    report "ERROR FATAL : Nsub < Ni or not defined"
      severity ERROR; --WARNING;
 END IF;
return model_Cellgamma;
END FUNCTION;
--*****************************************************************************************************
FUNCTION Ids_CURRENT(Vds,Vgs,Vbs,Von:VOLTAGE;
                                M_TEMP:TEMPERATURE ) return CURRENT IS

VARIABLE model_CelllatDiff :REAL:= LD;
VARIABLE here_Cell_W :REAL:= W;
VARIABLE here_Cell_L :REAL:= L;
```

```
VARIABLE model_CellLambda :REAL:= LAMBDA;
VARIABLE model_CelloxideThickness:REAL:=TOX;
VARIABLE model_CellsurfaceMobility :REAL:=U0;
VARIABLE model_CellMobilityTempCoeff :REAL:=BEX;
VARIABLE model_Celltnom             :TEMPERATURE:=TNOM;
VARIABLE here_Celltemp              :TEMPERATURE:=M_TEMP;
VARIABLE ratio,here_CelltSurfMob:REAL;

VARIABLE EffectiveLength,OxideCap, Beta1 :REAL;
VARIABLE model_Celltransconductance :REAL;
VARIABLE model_CelloxideCapFactor :REAL;
VARIABLE Vdsat,Vdsmin:VOLTAGE;
VARIABLE Ids:CURRENT;
BEGIN
  IF(model_Celltnom = UNDEF) THEN
    model_Celltnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Celltnom := model_Celltnom + CONSTCtoK;
  END IF;
  ratio := here_Celltemp/model_Celltnom;
  here_CelltSurfMob := model_CellsurfaceMobility * ratio**model_CellMobilityTempCoeff;

  model_CelloxideCapFactor := EPSOX/model_CelloxideThickness;
  model_Celltransconductance := here_CelltSurfMob * model_CelloxideCapFactor; --* 1.0e-4;
  EffectiveLength := here_Cell_L - (2.0 * model_CelllatDiff);
  OxideCap := model_CelloxideCapFactor * EffectiveLength * here_Cell_W;--COX'
  Beta1 := model_Celltransconductance * here_Cell_W/EffectiveLength;
 IF (Vgs < Von) THEN
    Ids :=0.0;
 ELSE
    Vdsat := (Vgs - Von);
    Vdsmin :=min(Vds,Vdsat);
    Ids :=   Beta1 * (1.0 + model_CellLambda * Vds ) * (Vgs - Von - (Vdsmin/2.0)) * Vdsmin;
 END IF;
return Ids;
END FUNCTION;
--*****************************************************************************
FUNCTION Itun_CURRENT(Vfgs,Vn2s:VOLTAGE) return CURRENT IS

VARIABLE model_CellAlphaEcr :REAL:=ALPHAECR;
VARIABLE model_CellAlphaEff :REAL:=ALPHAEFF;
VARIABLE model_CellBetaEcr :REAL:=BETAECR;
VARIABLE model_CellBetaEff :REAL:=BETAEFF;
VARIABLE model_CellTunelOxideThickness:REAL:=Ttun;
VARIABLE model_CellTunelArea          :REAL:=Stun;
--VARIABLE Etun, Alpha, Beta:REAL;
VARIABLE Etun:REAL;
VARIABLE Itun:CURRENT;
BEGIN
 Etun :=(Vfgs - Vn2s) /model_CellTunelOxideThickness;
 IF ((Vfgs - Vn2s)>0.0) THEN
   Itun := - model_CellTunelArea * model_CellAlphaEff * Etun**2 * exp(-model_CellBetaEff/Etun);
 ELSIF ((Vfgs - Vn2s)<0.0) THEN
   Itun := model_CellTunelArea * model_CellAlphaEcr * Etun**2 * exp(model_CellBetaEcr/Etun);
 ELSE
   Itun := IS_o;
 END IF;
RETURN Itun;
END FUNCTION;
```

```
--*****************************************************************************
 TERMINAL fg,n1,n2:electrical;
 CONSTANT model_Cellvt0                  :VOLTAGE:= VTO;
 CONSTANT model_CelllatDiff              :REAL:= LD;
 CONSTANT here_Cell_W                    :REAL:= W;
 CONSTANT here_Cell_L                    :REAL:= L;
 CONSTANT model_CelloxideThickness:REAL  :=TOX;
 CONSTANT model_CellfastSurfaceStateDensity  :REAL:=NFS;
 CONSTANT model_CellTunelArea            :REAL:=Stun;
 CONSTANT model_CellTunelOxideThickness  :REAL:=Ttun;
 CONSTANT model_CellInterPolyArea        :REAL:=Spp;
 CONSTANT model_CellInterPolyOxideThickness  :REAL:=Tpp;
 CONSTANT model_CellinitCharge           :CHARGE:= QFG0;

 QUANTITY EffectiveLength, model_CelloxideCapFactor, OxideCap:REAL;
 QUANTITY Von, Vtgc  :VOLTAGE;
 QUANTITY Keff, Kecr :REAL;
 QUANTITY vt :VOLTAGE;
 QUANTITY QPP,QTUN,QGD,QGS,QGB:CHARGE;
 QUANTITY Qfg : CHARGE;
 QUANTITY Etun :REAL;
 QUANTITY CPP, CGD, CGS, CGB, CTUN, Ctot :CAPACITANCE;

 QUANTITY Idbs  through Bulk to n1;
 QUANTITY Idbd  through Bulk to n2;
 QUANTITY Itun  through n2 to fg;
 QUANTITY Ids   through n2 To n1;

 QUANTITY Vn2  across   n2 TO bulk;
 QUANTITY Vn1  across   n1 TO bulk;
 QUANTITY Vgc  across   gc TO bulk;
 QUANTITY Vfg  across  Ivfg through  fg to Bulk;
 QUANTITY Vfgs across   fg to Source;
 QUANTITY Vds  across   Drain to Source;
-- Capacitance
 QUANTITY Vcpp  across Icpp  through gc TO fg;
 QUANTITY Vcgd  across Icgd  through n2 TO fg;
 QUANTITY Vctun across Ictun through n2 TO fg;
 QUANTITY Vcgb  across Icgb  through Bulk TO fg;
 QUANTITY Vcgs  across Icgs  through n1 TO fg;
 QUANTITY Vgcs  across              gc TO Source;
 QUANTITY Vn2s  across              n2 TO Source;
--  Resistance
 QUANTITY Vrd across Ird through Drain TO n2;
 QUANTITY Vrs across Irs through n1 TO Source;
 QUANTITY R_RD, R_RS :RESISTANCE;
 --Temperature
 constant CELL_Temp :TEMPERATURE:=defaulttemp;
-- QUANTITY CELL_Temp ACROSS T1 TO THERMAL_REF;
-- QUANTITY CELL_P THROUGH  T2 TO THERMAL_REF;
--QUANTITY CELL_Temp ACROSS  T1 TO THERMAL_REF;
BEGIN
  vt == CELL_Temp * CONSTKoverQ;
  model_CelloxideCapFactor == EPSOX/model_CelloxideThickness;
  EffectiveLength == here_Cell_L - (2.0 * model_CelllatDiff);
  OxideCap == model_CelloxideCapFactor * EffectiveLength * here_Cell_W;--COX'

  -- La tension de conduction Von
```

```
  Von == model_Cellvt0; --+ model_Cellgamma * (sqrt(PHI_FUN(CELL_TEMP + CONSTCtoK)  - (-Vn1)) -
sqrt(PHI_FUN(CELL_TEMP + CONSTCtoK)));

  CGB  == MEYER_C_CGB(Vfgs,Vds,Von,PHI_FUN(CELL_Temp + CONSTCtoK),OxideCap,
EffectiveLength);
  CGD  == MEYER_C_CGD(Vfgs,Vds,Von,PHI_FUN(CELL_Temp + CONSTCtoK),OxideCap,
here_Cell_W);
  CGS  == MEYER_C_CGS(Vfgs,Vds,Von,PHI_FUN(CELL_Temp + CONSTCtoK),OxideCap,
here_Cell_W);
  CTUN == TUNNEL_C_CTUN(Vfgs,Vds);
  CPP  == INTERPOLY_C_CPP(Vgcs,Vfgs);

  QPP  == CPP * (Vfg - Vgc);
  QTUN == CTUN * (Vfg - Vn2);
  QGD  == CGD * (Vfg - Vn2);
  QGS  == CGS * (Vfg - Vn1);
  QGB  == CGB * Vfg ;

  Icpp == QPP'dot;
  Ictun == QTUN'dot;
  Icgd == QGD'dot;
  Icgs == QGS'dot;
  Icgb == QGB'dot;

  Ctot == CTUN + CGB + CGS  + CPP + CGD;

  Keff == CPP/Ctot;
  Kecr == CTUN/Ctot;

  -- Tunnel Current
  Itun==Itun_CURRENT(Vfgs,Vds);
  Qfg == Itun'integ + model_CellinitCharge;
  Vfg == Keff*Vgc + Kecr*Vn2 + (Qfg + CGS*Vn1 + CGD*Vn2)/Ctot ;

  Vtgc == Von/Keff - Qfg/CPP;

  Etun ==(Vfgs - Vds)/model_CellTunelOxideThickness;

  Ids ==Ids_CURRENT(Vds,Vfgs,(-Vn1),Von,CELL_Temp + CONSTCtoK );

  Idbs == IBS_DIODE (-Vn1,CELL_Temp + CONSTCtoK );
  Idbd == IBS_DIODE (-Vn2,CELL_Temp + CONSTCtoK );

  -- RESISTANCE
  R_RS == R_RS_T(CELL_Temp + CONSTCtoK );
  R_RD == R_RD_T(CELL_Temp + CONSTCtoK );
  Vrs == Irs * R_RS;
  Vrd == Ird * R_RD;
--  CELL_P == Icpp * Vgc + Ids * Vds;
END Arch_de_Cell;
```

Annexe 3

MODÈLE VHDL-AMS D'UN TRANSISTOR MOS : UNICELL

```
LIBRARY IEEE; USE IEEE.math_REAL.all;
LIBRARY STD; USE STD.standard.all;
USE work.UnicellData.all;
USE work.electrical_systems.all;
USE work.thermal_systems.all;
ENTITY Unicell IS
  GENERIC (
        W           :REAL:=defaultMosW;
        L           :RELA:= defaultMosL;
        AD          :REAL:=0.0;
        AS          :REAL:=0.0;
        JS          :REAL:=0.0;
        IS_o        :REAL:=1.0e-14;
        MU0         :REAL:= 600.0;
        BEX         :REAL:=-1.5;
        GAMMA       :REAL :=UNDEF;
        PHI         :REAL :=UNDEF;
        NOX         :REAL :=0.0;
        TOX         :REAL :=1.0E-7;
        THETAG      :REAL := 0.0;
        THETAD      :REAL :=0.0;
        NJ          :REAL :=1.0e26;
        XJ          :REAL :=0.0;
        ETA         :REAL :=0.0;
        OPTEMP      :REAL:=1.0;
        LD          :REAL:=0.0;
        ALPHA       :REAL:=1.0;
        BETA        :REAL:=0.5;
        RD          :RESISTANCE:=UNDEF;
        RS          :RESISTANCE:=UNDEF;
        RSH         :RESISTANCE:=UNDEF;
        RDC         :RESISTANCE:=0.0;
        RSC         :RESISTANCE:=0.0;
        TRD1        :TEMPERATURE:=0.0;
        TRD2        :TEMPERATURE:=0.0;
        TRS1        :TEMPERATURE:=0.0;
        TRS2        :TEMPERATURE:=0.0;
        NFS         :REAL:=0.0;
        CGDO        :REAL:=0.0;
        CGSO        :REAL:=0.0;
```

```
      CGBO          :REAL:=0.0;
      NSUB          :REAL:=UNDEF;
      TNOM          :TEMPERATURE:=UNDEF);
PORT( terminal  drain,  gate,  source,  bulk  : electrical);
END ENTITY Unicell;

ARCHITECTURE arch_unicell OF Unicell IS
--*****************************************************************************************************************************
FUNCTION min(a,b:REAL) return REAL IS
BEGIN
  IF (a>b) THEN return b;
  ELSE return a;
  END IF;
END FUNCTION;
--*****************************************************************************************************************************
FUNCTION MEYER_C_CGB(Vd,Vg,Vs,Von:VOLTAGE; phi_1,cox_1:REAL; Leff:REAL
                                    ) return CAPACITANCE IS

VARIABLE Cgb, GateBulkOverlapCap :CAPACITANCE;
VARIABLE Vgs, Vds :VOLTAGE;
VARIABLE  a, b:REAL;
VARIABLE model_MOS3gateBulkOverlapCapFactor :REAL:= CGBO;
BEGIN
GateBulkOverlapCap := model_MOS3gateBulkOverlapCapFactor * Leff;
  Vds := Vd - Vs;
  Vgs := Vg - Vs;
  --Calcule de la capacite de CGB
  IF (Vgs < (Von - phi_1)) THEN                    -- Accumulation region
     Cgb := cox_1 + GateBulkOverlapCap;
  ELSIF (Vgs > (Von - phi_1)) and (Vgs < Von) THEN     -- Depletion region
--      Cgb := cox_1 *(1.0 +((2.0/3.0)*(Von - phi_1 - Vgs)/(phi_1))) + GateBulkOverlapCap;
     a:=2.0/(cox_1*phi_1);
     b:=(3.0 - 2.0*Von/phi_1)/cox_1;
     Cgb := 1.0/(a*Vgs+b) + GateBulkOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN     -- Saturation region
     Cgb := GateBulkOverlapCap;
  ELSE                                    -- Linear region
     Cgb := GateBulkOverlapCap;
  END IF;
return Cgb;
END FUNCTION;
--*****************************************************************************************************************************
FUNCTION MEYER_C_CGS(Vd,Vg,Vs,Von:VOLTAGE; phi_1, cox_1:REAL; Weff:REAL
                                    ) return CAPACITANCE IS

VARIABLE Cgs, GateSourceOverlapCap :CAPACITANCE;
VARIABLE Vds, Vgs :VOLTAGE;
VARIABLE model_MOS3gateSourceOverlapCapFactor :REAL:= CGSO;
BEGIN
  GateSourceOverlapCap := model_MOS3gateSourceOverlapCapFactor * Weff;
  Vds := Vd - Vs;
  Vgs := Vg - Vs;
  --Calcule de la capacite de CGS
  IF (Vgs < Von )  THEN                                    -- Accumulation region
     Cgs := GateSourceOverlapCap;
  ELSIF  (Vgs = Von) THEN                                  -- Depletion region
     Cgs := 2.0/3.0*cox_1*((Von - Vgs)/phi_1 + 1.0) + GateSourceOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN           -- Saturation region
     Cgs := 2.0/3.0*cox_1 + GateSourceOverlapCap ;
  ELSE                                                     -- Linear region
```

```
    Cgs := 2.0/3.0*cox_1*(1.0 - ( (Vgs - Vds - Von)/(2.0*(Vgs - Von) - Vds) )**2) + GateSourceOverlapCap;
  END IF;
return Cgs;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION MEYER_C_CGD(Vd,Vg,Vs,Von:VOLTAGE; phi_1, cox_1:REAL; Weff:REAL
                                    ) return CAPACITANCE IS

VARIABLE GateDrainOverlapCap, Cgd :CAPACITANCE;
VARIABLE Vds,Vgs:VOLTAGE;
VARIABLE model_MOS3gateDrainOverlapCapFactor :REAL:= CGDO;
BEGIN
  GateDrainOverlapCap := model_MOS3gateDrainOverlapCapFactor * Weff;
  Vds := Vd - Vs;
  Vgs := Vg - Vs;
  -- Calcule de la capacite de CGD
  IF (Vgs < (Von - phi_1)) THEN                                --Accumulation region
    Cgd := GateDrainOverlapCap;
  ELSIF (Vgs > (Von - phi_1)) and (Vgs < Von) THEN             --Depletion region
    Cgd := GateDrainOverlapCap;
  ELSIF (Vgs > Von) and (Vgs < (Von + Vds)) THEN               --Saturation region
    Cgd := GateDrainOverlapCap;
  ELSE                                                         --Linear region
    Cgd := 2.0/3.0*cox_1*(1.0 - ( (Vgs - Von)/(2.0*(Vgs - Von) - Vds) )**2) + GateDrainOverlapCap;
  END IF;
return Cgd;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION max(a,b:REAL) return REAL IS
BEGIN
  IF (a>b) THEN return a;
  ELSE return b;
  END IF;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION R_RD_T(defaultNRD,defaultM,R_TEMP:REAL ) return REAL IS

CONSTANT NRD :REAL:=defaultNRD;
CONSTANT M   :REAL:=defaultM;
VARIABLE delta_T, RDeff, arg, RD_T:REAL;
--*************************
--*  Device parameters *
--*************************
VARIABLE here_UNICELLdrainSquares :REAL:= NRD;
VARIABLE here_DraiContRes :REAL:= RDC;
VARIABLE here_NumParaTra  :REAL:= M;
--************************
--*  Model parameters *
--************************
VARIABLE model_Rtnom :REAL:= TNOM;
VARIABLE model_RtempCoeffLinearRD     :REAL:= TRD1;
VARIABLE model_RtempCoeffQuadraticRD  :REAL:= TRD2;
VARIABLE model_UNICELLsheetResistance    :REAL:= RSH;
VARIABLE model_UNICELLdrainResistance    :REAL:= RD;
BEGIN
  IF(model_Rtnom = UNDEF) THEN
    model_Rtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Rtnom := model_Rtnom + CONSTCtoK;
  END IF;
```

```
  delta_T := R_TEMP - model_Rtnom;
  arg := 1.0 + model_RtempCoeffLinearRD  * delta_T + model_RtempCoeffQuadraticRD  * delta_T**2;

  IF(model_UNICELLdrainResistance /= UNDEF) THEN
    IF(model_UNICELLdrainResistance /= 0.0) THEN
      model_UNICELLdrainResistance := RD;
    ELSE
      model_UNICELLdrainResistance := 0.0;
    END IF;
  ELSIF (model_UNICELLsheetResistance /= UNDEF) THEN
    IF(model_UNICELLsheetResistance /= 0.0) THEN
       IF (here_UNICELLdrainSquares > 0.0) THEN
          model_UNICELLdrainResistance := model_UNICELLsheetResistance*here_UNICELLdrainSquares;
       ELSE
         REPORT "ERROR FATAL: Squares of drain resistance <= 0.0 !!!"
         SEVERITY ERROR; --WARNING;
       END IF;
    ELSE
      model_UNICELLdrainResistance := 0.0;
    END IF;
  ELSE
    model_UNICELLdrainResistance:= 0.0;
  END IF;

  RDeff := ( model_UNICELLdrainResistance + here_DraiContRes) / here_NumParaTra;

  RD_T := RDeff * arg;
  return RD_T;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION R_RS_T(defaultNRS,defaultM :REAL; R_TEMP :TEMPERATURE ) return REAL IS

constant NRS :REAL:=defaultNRS;
constant M   :REAL:=defaultM;
VARIABLE delta_T, RSeff, arg, RS_T:REAL;
--**************************
--*  Device parameters *
--**************************
VARIABLE here_UNICELLsourceSquares :REAL:= NRS;
VARIABLE here_SourContRes :REAL:= RSC;
VARIABLE here_NumParaTra  :REAL:=M;
--*************************
--*  Model parameters *
--*************************
VARIABLE model_Rtnom :REAL:= TNOM;
VARIABLE model_RtempCoeffLinearRS     :REAL:=TRS1;
VARIABLE model_RtempCoeffQuadraticRS  :REAL:=TRS2;
VARIABLE model_UNICELLsheetResistance    :REAL:= RSH;
VARIABLE model_UNICELLsourceResistance   :REAL:= RS;
BEGIN
  IF(model_Rtnom = UNDEF) THEN
    model_Rtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_Rtnom := model_Rtnom + CONSTCtoK;
  END IF;

  delta_T := R_TEMP - model_Rtnom;
  arg := 1.0 + model_RtempCoeffLinearRS  * delta_T + model_RtempCoeffQuadraticRS  * delta_T**2;
  IF(model_UNICELLsourceResistance /= UNDEF ) THEN
```

```
    IF(model_UNICELLsourceResistance /= 0.0) THEN
      model_UNICELLsourceResistance := RS;
    ELSE
      model_UNICELLsourceResistance := 0.0;
    END IF;
  ELSIF (model_UNICELLsheetResistance /= UNDEF) THEN
    IF(model_UNICELLsheetResistance /= 0.0) THEN
      IF (here_UNICELLsourceSquares > 0.0) THEN
        model_UNICELLsourceResistance :=
model_UNICELLsheetResistance*here_UNICELLsourceSquares;
      ELSE
        REPORT "ERROR FATAL: Squares of source resistance <= 0.0 !!!"
        SEVERITY ERROR; --WARNING;
      END IF;
    ELSE
      model_UNICELLsourceResistance:= 0.0;
    END IF;
  ELSE
    model_UNICELLsourceResistance := 0.0;
  END IF;
  RSeff := (model_UNICELLsourceResistance + here_SourContRes) / here_NumParaTra;
  RS_T := RSeff * arg;
  return RS_T;
END FUNCTION;
--*****************************************************************************************************************************
FUNCTION IBS_DIODE (Vbs :VOLTAGE; D_TEMP:TEMPERATURE ) RETURN CURRENT IS

VARIABLE model_UNICELLtnom :REAL:=TNOM;
VARIABLE model_UNICELLjctSatCur     :CURRENT:=IS_o;
VARIABLE model_UNICELLjctSatCurDensity:REAL:=JS;

VARIABLE here_UNICELLgbs   :REAL;
VARIABLE here_UNICELLcbs_i :CURRENT;
VARIABLE here_UNICELLtemp  :REAL:=D_TEMP;
VARIABLE here_UNICELLsourceArea :REAL:=AS;

VARIABLE here_UNICELLtSatCurDens :REAL;
VARIABLE here_UNICELLtSatCur:REAL;

VARIABLE SourceSatCur,evbs,vtnom,vt,egfet1,egfet :REAL;
BEGIN
  IF(model_UNICELLtnom = UNDEF) THEN
    model_UNICELLtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_UNICELLtnom := model_UNICELLtnom + CONSTCtoK;
  END IF;
  IF(here_UNICELLtemp = UNDEF) THEN here_UNICELLtemp := defaulttemp + CONSTCtoK; END IF;

  vtnom := model_UNICELLtnom * CONSTKoverQ;
  vt := here_UNICELLtemp * CONSTKoverQ;
  egfet1 := 1.16-(7.02e-4*model_UNICELLtnom*model_UNICELLtnom)/(model_UNICELLtnom+1108.0);
  egfet := 1.16-(7.02e-4*here_UNICELLtemp*here_UNICELLtemp)/(here_UNICELLtemp+1108.0);
  here_UNICELLtSatCur := model_UNICELLjctSatCur * exp(-egfet/vt+egfet1/vtnom);
  here_UNICELLtSatCurDens := model_UNICELLjctSatCurDensity * exp(-egfet/vt+egfet1/vtnom);

  IF( (here_UNICELLtSatCurDens = 0.0) or (here_UNICELLsourceArea = 0.0)) THEN
    SourceSatCur := here_UNICELLtSatCur;
  ELSE
    SourceSatCur := here_UNICELLtSatCurDens * here_UNICELLsourceArea;
  END IF;
```

```
  IF(Vbs <= 0.0) THEN
    here_UNICELLgbs := SourceSatCur/vt;
    here_UNICELLcbs_i := here_UNICELLgbs*Vbs;  --Bulk-Source Current
    here_UNICELLgbs :=  here_UNICELLgbs + ckt_CKTgmin;
  ELSE
    evbs := exp(MIN(MAX_EXP_ARG,Vbs/vt));
    here_UNICELLgbs := SourceSatCur*evbs/vt + ckt_CKTgmin;
    here_UNICELLcbs_i := SourceSatCur * (evbs-1.0);
  END IF;
  return here_UNICELLcbs_i;
END FUNCTION;
--*****************************************************************************************************************************
FUNCTION IBD_DIODE (Vbd :VOLTAGE; D_TEMP:TEMPERATURE )RETURN CURRENT IS

VARIABLE model_UNICELLtnom :REAL:=TNOM;
VARIABLE model_UNICELLjctSatCur     :CURRENT:=IS_o;
VARIABLE model_UNICELLjctSatCurDensity:REAL:=JS;
VARIABLE here_UNICELLgbd   :REAL;
VARIABLE here_UNICELLcbd_i :CURRENT;
VARIABLE here_UNICELLtemp  :REAL:=D_TEMP;
VARIABLE here_UNICELLdrainArea :REAL:=AD;
VARIABLE here_UNICELLtSatCurDens :REAL;
VARIABLE here_UNICELLtSatCur:REAL;
VARIABLE evbd,DrainSatCur,vtnom,vt,egfet1,egfet :REAL;
BEGIN
  IF(model_UNICELLtnom = UNDEF) THEN
    model_UNICELLtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_UNICELLtnom := model_UNICELLtnom + CONSTCtoK;
  END IF;
IF(here_UNICELLtemp = UNDEF) THEN here_UNICELLtemp := defaulttemp + CONSTCtoK; END IF;

  vtnom := model_UNICELLtnom * CONSTKoverQ;
  vt := here_UNICELLtemp * CONSTKoverQ;
  egfet1 := 1.16-(7.02e-4*model_UNICELLtnom*model_UNICELLtnom)/(model_UNICELLtnom+1108.0);
  egfet := 1.16-(7.02e-4*here_UNICELLtemp*here_UNICELLtemp)/(here_UNICELLtemp+1108.0);
  here_UNICELLtSatCur := model_UNICELLjctSatCur * exp(-egfet/vt+egfet1/vtnom);
  here_UNICELLtSatCurDens := model_UNICELLjctSatCurDensity * exp(-egfet/vt+egfet1/vtnom);

  IF( (here_UNICELLtSatCurDens = 0.0) or (here_UNICELLdrainArea = 0.0)) THEN
    DrainSatCur := here_UNICELLtSatCur;
  ELSE
    DrainSatCur := here_UNICELLtSatCurDens * here_UNICELLdrainArea;
  END IF;

  IF(Vbd <= 0.0) THEN
    here_UNICELLgbd := DrainSatCur/vt;
    here_UNICELLcbd_i := here_UNICELLgbd*Vbd;  --Bulk-Drain Current
    here_UNICELLgbd :=  here_UNICELLgbd + ckt_CKTgmin;
  ELSE
    evbd := exp(MIN(MAX_EXP_ARG,Vbd/vt));
    here_UNICELLgbd := DrainSatCur   * evbd/vt + ckt_CKTgmin;
    here_UNICELLcbd_i := DrainSatCur * (evbd-1.0);
  END IF;
  return  here_UNICELLcbd_i;
END FUNCTION;
--*****************************************************************************************************************************
FUNCTION Eg_T(T:TEMPERATURE ) return VOLTAGE IS
VARIABLE model_UNICELLoptemp          :REAL:=OPTEMP;
```

```
VARIABLE Eg :VOLTAGE;
BEGIN
 IF (model_UNICELLoptemp = 1.0) THEN
   Eg := 1.16 - 7.02e-4 * T**2/(T + 1108.0);
 ELSIF (model_UNICELLoptemp = 2.0) THEN
   Eg := 1.17 - 4.73e-4 * T**2/(T + 636.0);
 ELSIF (model_UNICELLoptemp = 3.0) THEN
   IF (T <= 170.0) THEN
     Eg := 1.17 + T * 1.059e-5 - 6.05e-7 * T**2;
   ELSE
     Eg := 1.1785 + T * 9.025e-5 - 3.05e-7 * T**2;
   END IF;
 ELSE
   report "ERROR FATAL : OPTEMP must be in {1,2,3}"
     severity ERROR; --WARNING;
 END IF;
return Eg;
END FUNCTION;
--*****************************************************************************************************************
FUNCTION Ni_T(T, TNOM:TEMPERATURE; ut :VOLTAGE; Eg :REAL) return REAL IS

VARIABLE ni :REAL;
BEGIN
  ni:=1.45e16*(T/TNOM)**(3.0/2.0)*exp(Eg*(T/TNOM - 1.0)*(1.0/(2.0*ut)));
  return ni;
END FUNCTION;
--*****************************************************************************************************************
FUNCTION PSI_Fon(Vg,V,ut,Vfb:VOLTAGE; psi:REAL) return REAL IS

 VARIABLE model_UNICELLgamma          :REAL:=GAMMA;
 VARIABLE model_UNICELLphi            :REAL:= PHI;
 VARIABLE psi1:REAL;
BEGIN
  psi1:= Vg-Vfb-model_UNICELLgamma*(psi+ut*(exp((psi-model_UNICELLphi-V)/ut)-1.0))**0.5;
 return psi1;
END FUNCTION;
--*****************************************************************************************************************
FUNCTION PHI_FUN(PHI_TEMP :TEMPERATURE ) RETURN REAL IS

VARIABLE model_UNICELLphi            :VOLTAGE:=PHI;
VARIABLE model_UNICELLsubstrateDoping :REAL:=NSUB;
VARIABLE model_UNICELLtnom           :TEMPERATURE:=TNOM;
VARIABLE here_UNICELLtemp            :TEMPERATURE:=PHI_TEMP;
VARIABLE here_UNICELLtPhi            :REAL;
VARIABLE vt,ratio, fact2, kt, egfet, arg, pbfact, vtnom, phio :REAL;
VARIABLE fact1, kt1, egfet1, arg1, pbfact1 :REAL;
BEGIN
  IF(model_UNICELLtnom = UNDEF) THEN
    model_UNICELLtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_UNICELLtnom := model_UNICELLtnom + CONSTCtoK;
  END IF;
  vtnom := model_UNICELLtnom*CONSTKoverQ;
  IF( model_UNICELLsubstrateDoping/= UNDEF) THEN
    IF(model_UNICELLsubstrateDoping*1.0e6 > 1.45e16) THEN --/*(cm**3/m**3)*/
      IF(model_UNICELLphi = UNDEF) THEN
        model_UNICELLphi := 2.0*vtnom*log(model_UNICELLsubstrateDoping*1.0e6/1.45e16);--
(cm**3/m**3)
        model_UNICELLphi := max(0.1,model_UNICELLphi);
      END IF;
```

```
    END IF;
  ELSE
     model_UNICELLsubstrateDoping := 0.0;
     report "ERROR FATAL : Nsub < Ni or not defined"
        severity ERROR; --WARNING;
  END IF;

  fact1 := model_UNICELLtnom/REFTEMP;
  kt1 := CONSTboltz * model_UNICELLtnom;
  egfet1 := 1.16-(7.02e-4*model_UNICELLtnom*model_UNICELLtnom)/(model_UNICELLtnom+1108.0);
  arg1 := -egfet1/(kt1+kt1)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
  pbfact1 := -2.0*vtnom *(1.5*log(fact1)+q*arg1);

  vt := here_UNICELLtemp * CONSTKoverQ;
  ratio := here_UNICELLtemp/model_UNICELLtnom;
  fact2 := here_UNICELLtemp/REFTEMP;
  kt := here_UNICELLtemp * CONSTboltz;
  egfet := 1.16-(7.02e-4*here_UNICELLtemp*here_UNICELLtemp)/(here_UNICELLtemp+1108.0);
  arg := -egfet/(kt+kt)+1.1150877/(CONSTboltz*(REFTEMP+REFTEMP));
  pbfact := -2.0*vt *(1.5*log(fact2)+q*arg);
  phio := (model_UNICELLphi-pbfact1)/fact1;
  here_UNICELLtPhi := fact2 * phio + pbfact;
  RETURN here_UNICELLtPhi;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION GAMMA_FUN RETURN REAL IS

VARIABLE model_UNICELLgamma          :REAL:=GAMMA;
VARIABLE model_UNICELLsubstrateDoping :REAL:=NSUB;
VARIABLE model_UNICELLoxideThickness     :REAL:=TOX;
VARIABLE model_UNICELLoxideCapFactor     :REAL;
BEGIN
 IF( model_UNICELLsubstrateDoping/= UNDEF) THEN
   IF(model_UNICELLsubstrateDoping*1.0e6 > 1.45e16) THEN --/*(cm**3/m**3)*/
      IF(model_UNICELLgamma = UNDEF) THEN
         model_UNICELLoxideCapFactor := 3.9 * 8.854214871e-12/model_UNICELLoxideThickness;
         model_UNICELLgamma := sqrt(2.0 * EPSSIL * q * model_UNICELLsubstrateDoping*1.0e6)/
model_UNICELLoxideCapFactor;--/*(cm**3/m**3)*/
      END IF;
    END IF;
  ELSE
     model_UNICELLsubstrateDoping := 0.0;
     report "ERROR FATAL : Nsub < Ni or not defined"
        severity ERROR; --WARNING;
  END IF;
return model_UNICELLgamma;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION MOBILITY_FUN(M_TEMP :TEMPERATURE) RETURN REAL IS

VARIABLE model_UNICELLsurfaceMobility   :REAL:= MU0*1.0e-4;
VARIABLE model_UNICELLMobilityTempCoeff :REAL:=BEX;
VARIABLE model_UNICELLtnom              :TEMPERATURE:=TNOM;
VARIABLE here_UNICELLtemp              :TEMPERATURE:=M_TEMP;
VARIABLE ratio,here_UNICELLtSurfMob:REAL;
BEGIN
  IF(model_UNICELLtnom = UNDEF) THEN
    model_UNICELLtnom := defaulttnom + CONSTCtoK;
  ELSE
    model_UNICELLtnom := model_UNICELLtnom + CONSTCtoK;
```

```
  END IF;
 ratio := here_UNICELLtemp/model_UNICELLtnom;
 here_UNICELLtSurfMob := model_UNICELLsurfaceMobility * ratio**model_UNICELLMobilityTempCoeff;
 RETURN here_UNICELLtSurfMob;
END FUNCTION;
--*************************************************************************************************************************
FUNCTION TNOM_FUN(TNOM:TEMPERATURE) RETURN TEMPERATURE IS

VARIABLE UNICELL_tnom :TEMPERATURE;
BEGIN
  IF(TNOM = UNDEF) THEN
    UNICELL_tnom := defaulttnom + CONSTCtoK;
  ELSE
    UNICELL_tnom := UNICELL_tnom + CONSTCtoK;
  END IF;
return UNICELL_tnom;
END FUNCTION;
--*************************************************************************************************************************
CONSTANT Na :REAL:=1.0e22;
QUANTITY model_UNICELLoxideCapFact    :REAL;
--QUANTITY model_UNICELLphi            :REAL;
--CONSTANT model_UNICELLsurfaceMobility :REAL:= MU0*1.0e-4;
CONSTANT model_UNICELLdenChargeFix    :REAL:=NOX*1.0e4;
CONSTANT model_UNICELLoxideThickness  :REAL:= TOX;
CONSTANT model_UNICELLtnom            :TEMPERATURE:=TNOM;
CONSTANT model_UNICELLthetad          :REAL:= THETAD;
CONSTANT model_UNICELLthetag          :REAL:= THETAG;
CONSTANT model_UNICELLalpha           :REAL:=ALPHA;
CONSTANT model_UNICELLbeta            :REAL:=BETA;
CONSTANT model_UNICELLdopageSouDra    :REAL:= NJ;
CONSTANT model_UNICELLeta             :REAL:=ETA;
CONSTANT model_UNICELLxj              :REAL:=XJ;
CONSTANT model_UNICELLlatDiff         :REAL:= LD;
CONSTANT here_UNICELL_W               :REAL:= W;
CONSTANT here_UNICELL_L               :REAL:= L;

QUANTITY ut, Vfb:VOLTAGE;
QUANTITY OxideCap,alpha1, alpha2, alpha3:REAL;
QUANTITY psis :REAL:=1.0;
QUANTITY psid :REAL:=1.0;
QUANTITY Eg :REAL;
QUANTITY Vbi, Von, Vdsat :VOLTAGE;
QUANTITY mu, ksi1, ksi2 :REAL; -- Mobilite
QUANTITY DomainType :REAL:= 0.0;
QUANTITY UNICELL :TEMPERATURE;
QUANTITY T, mp, mn, sqrt_nc_nv,eee :REAL;
QUANTITY x0,  ys, yd, NI :REAL;
QUANTITY x0, xs, xd, wjs, wjd, Ni, ys, yd, Lc,EffectiveLength  :REAL; -- Modulation de longueur du canal
terminal s1, d1:Electrical;
QUANTITY Vg across Gate to Bulk;
QUANTITY Vs across S1 to Bulk;
QUANTITY Vd across D1 to Bulk;
QUANTITY Id through D1 to S1;
QUANTITY Idq through Drain;
-- Courant diode
QUANTITY VDbd across IDbd through Bulk to D1;
QUANTITY VDbs across IDbs through Bulk to S1;
-- Resistant
QUANTITY Vrd across Idq through Drain to D1;
QUANTITY Vrs across Isq through S1 to Source;
```

```
QUANTITY R_RS, R_RD :RESISTANCE;
-- Capacitance
QUANTITY Vcgd across Icgd through Gate to D1;
QUANTITY Vcgs across Icgs through Gate to S1;
QUANTITY Vcgb across Icgb through Gate to Bulk;
QUANTITY C_CGD, C_CGS, C_CGB :CAPACITANCE;
-- TEMP
-- QUANTITY UNICELL_Temp ACROSS UNICELL_P THROUGH T1 TO THERMAL_REF;
CONSTANT UNICELL_temp           :TEMPERATURE:=defaulttemp;
BEGIN
  UNICELL== MOBILITY_FUN(UNICELL_temp + CONSTCtoK);
  model_UNICELLgamma == GAMMA_FUN;
  model_UNICELLphi==PHI_FUN(UNICELL_temp + CONSTCtoK );
  UNICELL_tnom==TNOM_FUN(model_UNICELLtnom);
  Eg ==EG_T(TNOM_FUN(model_UNICELLtnom));
  ut ==(UNICELL_temp + CONSTCtoK) * CONSTKoverQ;
  model_UNICELLoxideCapFact == EPSOX /model_UNICELLoxideThickness;
  Vfb ==-0.5 *( Eg + PHI_FUN(UNICELL_temp + CONSTCtoK ))+ q
*model_UNICELLdenChargeFix/model_UNICELLoxideCapFact;
  Xn == model_UNICELLeta + q*NFS/OxideCap;
  psis    == Vg-Vfb-GAMMA_FUN*(psis+ut*(exp((psis-PHI_FUN(UNICELL_temp + CONSTCtoK )-Vs)/(ut))-
1.0))**0.5;
  psid    == Vg-Vfb-GAMMA_FUN*(psid+ut*(exp((psid-PHI_FUN(UNICELL_temp + CONSTCtoK )-
Vd)/(ut))-1.0))**0.5;
  -- Correction de mobilite
  ksi1    == Vg-Vfb-0.5*(psis+psid);
  ksi2    == 0.66*GAMMA_FUN*(((psid-ut)**1.5-(psis-ut)**1.5)/(psid-psis+1.0e-6));
  MU      == MOBILITY_FUN(UNICELL_temp + CONSTCtoK)/(1.0 + model_UNICELLthetad*(psid-psis)+
model_UNICELLthetag*(ksi1+ksi2));
  -- Modulation de la longueur du canal
  Ni == Ni_T(UNICELL_temp + CONSTCtoK ,TNOM_FUN(model_UNICELLtnom),ut,Eg);

  x0 == (2.0*EPSSIL/( q * Na ) )**0.5;
  Vbi== ut*log((Na*model_UNICELLdopageSouDra)/Ni**2);
  ys == x0*(abs(Vbi + Vs - psis))**0.5;
  yd == x0*(abs(Vbi + Vd - psid))**0.5;
  xs == x0*(psis-ut)**0.5;
  xd == x0*(psid-ut)**0.5;
  wjs == x0*(vbi+vs)**0.5;
  wjd == x0*(vbi+vd)**0.5;
  Lc  == here_UNICELL_L + 2.0*model_UNICELLeta*model_UNICELLxj-
(model_UNICELLeta*model_UNICELLxj+yd)*(1.0-(xd/(model_UNICELLxj+wjd))**model_UNICELLalpha)
        - (model_UNICELLeta*model_UNICELLxj+ys)*(1.0-
(xs/(model_UNICELLxj+wjs))**model_UNICELLalpha);
  EffectiveLength == Lc - 2.0*model_UNICELLlatDiff;
  -- Drain Current
  alpha1 == (Vg-Vfb+ut)*(psid-psis)-0.5*(psid*psid-psis*psis);
  alpha2 == GAMMA_FUN*0.666*(((psid-ut)**1.5)-((psis-ut)**1.5));
  alpha3 == GAMMA_FUN*ut*((psid-ut)**0.5-(psis-ut)**0.5);
  Id == (MU*here_UNICELL_W*model_UNICELLoxideCapFact/EffectiveLength)*(alpha1-alpha2+alpha3);
  -- tension seuil
  Von == Vfb + psis -0.5*GAMMA_FUN**2 + GAMMA_FUN * sqrt(Vg - Vfb - ut + 0.25*GAMMA_FUN**2);
  Vdsat == Vg - Von;
  -- Diode Dbd et Dbs
  IDbd ==IBD_DIODE (VDbd,UNICELL_Temp + CONSTCtoK);
  IDbs ==IBS_DIODE (VDbs,UNICELL_Temp + CONSTCtoK);
  -- Capacitance
   OxideCap == model_UNICELLoxideCapFact * EffectiveLength * here_UNICELL_W;
  IF (DOMAIN = TIME_DOMAIN) or (DOMAIN = FREQUENCY_DOMAIN) USE
    -- TIME DOMAIN and FREQUENCY DOMAIN
```

```
    DomainType == 1.0;
    -- Capacitance
    C_CGD == MEYER_C_CGD(Vd,Vg,Vs,Von,PHI_FUN(UNICELL_temp + CONSTCtoK ), OxideCap,
here_UNICELL_W);
    C_CGS == MEYER_C_CGS(Vd,Vg,Vs,Von,PHI_FUN(UNICELL_temp + CONSTCtoK ), OxideCap,
here_UNICELL_W);
    C_CGB == MEYER_C_CGB(Vd,Vg,Vs,Von,PHI_FUN(UNICELL_temp + CONSTCtoK ), OxideCap,
EffectiveLength);

    Icgd == C_CGD * Vcgd'dot;
    Icgs == C_CGS * Vcgs'dot;
    Icgb == C_CGB * Vcgb'dot;
  ELSE
    -- QUIESCENT DOMAIN
    DomainType == 2.0;
    -- Capacitance
    C_CGD == 0.0;
    C_CGS == 0.0;
    C_CGB == 0.0;
    Icgd == 0.0;
    Icgs == 0.0;
    Icgb == 0.0;
  END USE;

  -- Resistance
  R_RS == R_RS_T(defaultNRS,defaultM,UNICELL_Temp + CONSTCtoK );
  R_RD == R_RD_T(defaultNRD,defaultM,UNICELL_Temp + CONSTCtoK );
  Vrd == Idq * R_RD;
  Vrs == Isq * R_RS;
END arch_unicell;
```

Annexe 4

MACRO-MODELE D'UN AMPLIFICATEUR OPERATIONEL MODE TENSION (AOV) MODELISE PAR PSPICE

Paramètre générique du modèle

Le macro-modèle de cet amplificateur est caractérisé par une réponse en fréquence du deuxième ordre paramétrée par (Figure 1)(Tableau 1) :

Paramètre	Description	Unité	Valeur de test
ADC	Gain fini	dB	60.0
FT	Fréquence de transition	Hz	1.0Meg
MP	Marge de phase	°	76.0
RRMC	Taux de réjection de mode commun (Commun Mode Rejection Ratio)	dB	100.0
PSRRP	Taux de réjection de l'alimentation pôle positif (Power Supply rejection ratio).	dB	100.0
PSRRN	Taux de réjection de l'alimentation pôle négatif (Power Supply rejection ratio).	dB	100.0
SR	Slew Rate	V/µs	2.0
VOS	Tension de décalage (Offset)	V	1.0m
IB	Courant de polarisation	A	80.0n
IOS	Courant de décalage	A	20.0n
IMAXSO	Courant maximal qui peut fournir l'AOP	A	25.0n
IMAXSI	Courant maximal qui peut recevoir l'AOP	A	25.0n
RIND	Résistance d'entrée différentielle	Ω	100.0Meg
RINCM	Résistance d'entrée en mode commun	Ω	2.0G
ROUT	Résistance de sortie	Ω	1.0k
CIND	Capacité d'entrée différentielle	F	1.0p
CINCM	Capacité d'entrée en mode commun	F	0.1p

Tableau 6 : Les paramètres de l'AOV.

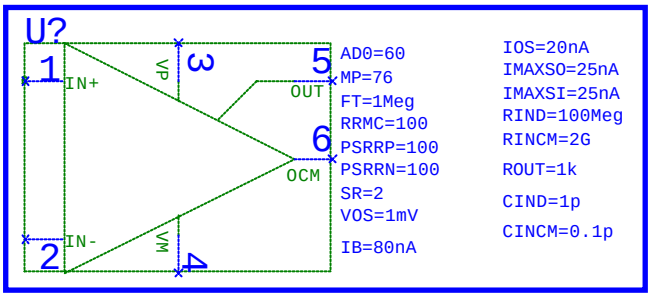


Figure 19 : Icône de l'amplificateur opérationnel et les paramètres génériques.

Macro-modèlisation de l'AOV

Le macro-modèle est divisé en cinq étages indépendant (isolé par les sources contrôlée): étage d'entrée, étage de gain, réponse en fréquence, slew rate, étage de sortie (Figure 2).

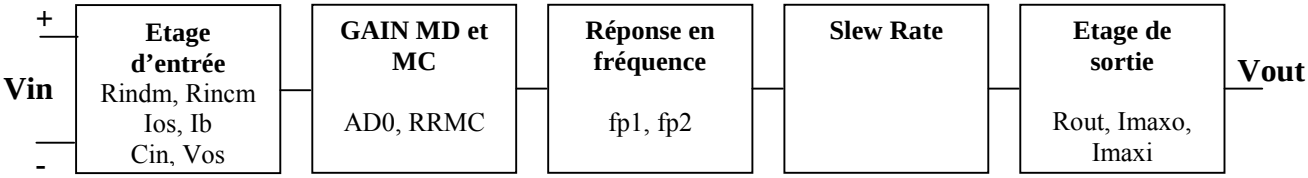


Figure 20 : Bloc fonctionnel de l'amplificateur.

Etage d'entrée

Les éléments primitif de SPICE qui forme l'étage d'entrée sont montrés à la figure 3. Le modèle de cet étage est constitué par des résistances et des capacités d'entrée de mode commun et de mode différentiel, une tension d'*offset*, courant de polarisation (*bias*), et courant d'*offset*.

Les résistances (capacités) de mode commun et de mode différentiel sont modélisés par R_{incm1}, R_{incm2} et R_{ind} (C_{incm1}, C_{incm2} et C_{ind}).

$$R_{incm1} = R_{incm2} = RINCM \quad (16)$$

$$R_{ind} = RIND \quad (17)$$

$$C_{incm1} = C_{incm2} = CINCM \quad (18)$$

$$C_{ind} = CIND \quad (19)$$

La tension d'offset est modélisée par V_{os}.

$$V_{os} = VOS \quad (20)$$

Les courants de polarisation et d'offset sont modélisés par I_{b1} et I_{b2}.

$$I_{b1} = IB + IOS/2 \quad (21)$$

$$I_{b2} = IB - IOS/2 \quad (22)$$

Gain en mode commun et en mode différentiel

Cet étage est constitué de deux sources dépendant de type VCCS (Figure 3). Les branchements de ces sources fait qu'il favorise un chemin et il ignore l'autre. Si l'AOV est branché en mode commun donc là, nous favorisons le GMC, si le cas contraire, nous favorisons GMD1.

$$GMC=(V(IN+,0)+V(IN-,0)).\frac{1}{2.gain\_cm0} \quad (23)$$

$$GMD1=V(IN+,IN-).gm1 \quad (24)$$

Avec gain_cm0 = $10^{RRMC/20}$ et gm1 = 1

Réponse en fréquence [POR03]

La tension de sortie référencé par rapport à la tension de sortie de mode commun OCM est de la forme :

$$V_s=\frac{gm1}{gd1}.\frac{gm2}{gd2}.\frac{V_{EP}-V_{EM}}{(1+\frac{C_1}{gd1}.p).(1+\frac{C_2}{gd2}.p)} \quad (25)$$

Devient :

$$V_s=AD0.\frac{V_{EP}-V_{EM}}{(1+\frac{1}{\omega 1}.p).(1+\frac{1}{\omega 2}.p)} \quad (26)$$

A partir de l'expression de la marge de phase, nous posons :

$$MP\approx\frac{\omega 2}{\omega T} \quad (27)$$

Et partir de la définition de la fréquence de transition (F_T), nous posons :

$$\frac{AD0}{\sqrt{(1+\frac{\omega_T^2}{\omega_1^2}).(1+\frac{\omega_T^2}{\omega_2^2})}}=1 \quad (28)$$

A partir des équations (10) au (13), nous déterminons les valeurs des capacités (C1 et C2) et des résistances (R1 et R2) :

$$C1=\frac{gd1}{\frac{2.\pi.F_T}{AD0}.\sqrt{1+\frac{F_T^2}{F_2^2}}} \quad (29)$$

$$C2=\frac{gd2}{2.\pi.F_T.\tan(MP)} \quad (30)$$

$$R1=\frac{1}{gd1} \text{ et } R2=\frac{1}{gd2} \quad (31) \text{ et } (32)$$

Avec $gd1=\frac{1}{10^{\frac{AD0}{20}}}$ et $gd2=1$.

Nous posons aussi $gm1=gm2=1$.

Slew Rate

Cet étage limite le courant dans la capacité C_2 pour produire le slew rate désiré. F1 est une source polynomial qui sert à donner un sens de courant dans la capacité. Avec les diodes D5 et D6 (D7 et D8), nous programmons le sens courant dans la capacité C_2 pour le slew rate positif (pour le slew rate négatif). Pour rendre les diodes linaire, nous choissions N = 0.001 (coefficient d'émission) et IS = 1fA (courant de saturation) (Voir chapitre 2).

$$I_{maxp} = I_{maxm} = C_2 .SR \tag{33}$$

Etage de sortie

Cet étage est constitué par une source dépendante (VCCS) G3, par une résistance de sortie R_{out}, et les courants de sortie/entrée qui limite la caractéristique de l'amplificateur (Imaxso et Imaxsi) (Figrue 9). Avec les diodes (D1, D2, D3, et D4), nous détectons les limites de saturation de l'amplificateur. De même, les diodes sont idéaux. Les équation élémentaire de chaque composant est de la forme :

$$G3 = \frac{V(IN+,IN-)}{R_{out}} \tag{34}$$

$$R_{out} = ROUT \tag{35}$$

$$I_{maxso} = IMAXSO \text{ et } I_{maxsi} = IMAXSI \tag{36}$$

Test du macro-modèlisation de l'AOV

Voir les figures de 4 au 14 ci-dessous.

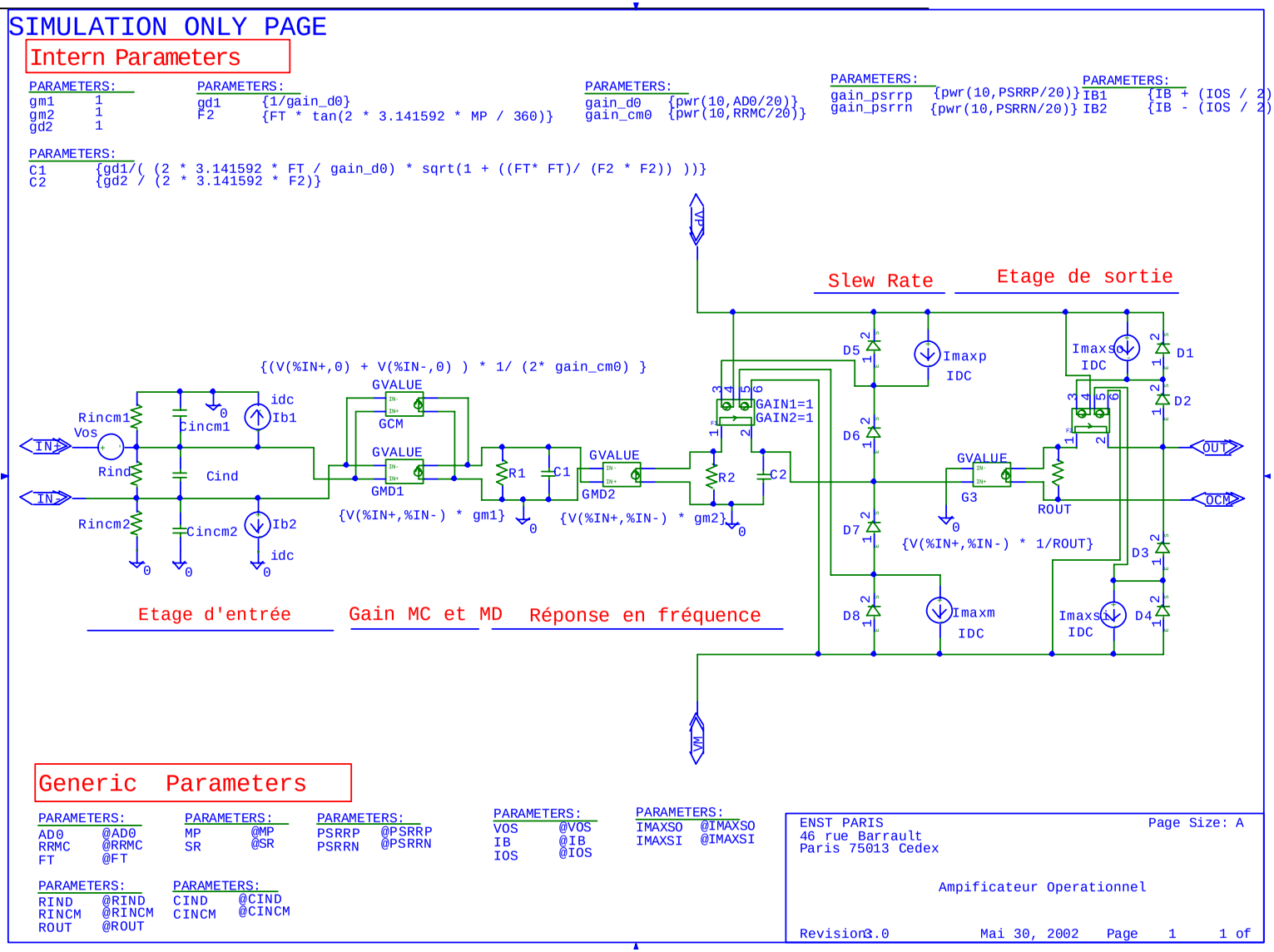


Figure 21 : Macro-modèle d'un amplificateur opérationnel modélisé par PSPICE.

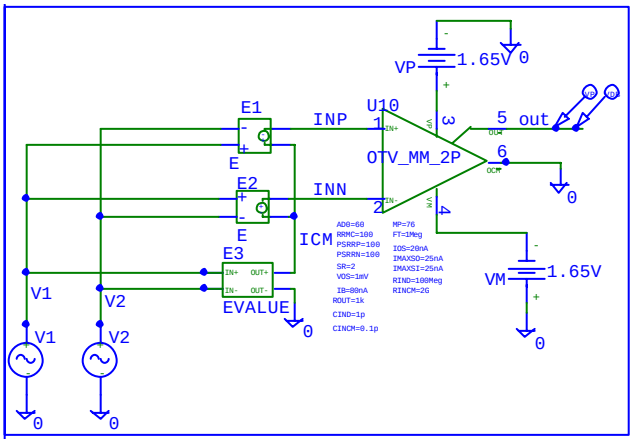


Figure 22 : Test en AC du mode commun et le mode différentielle.

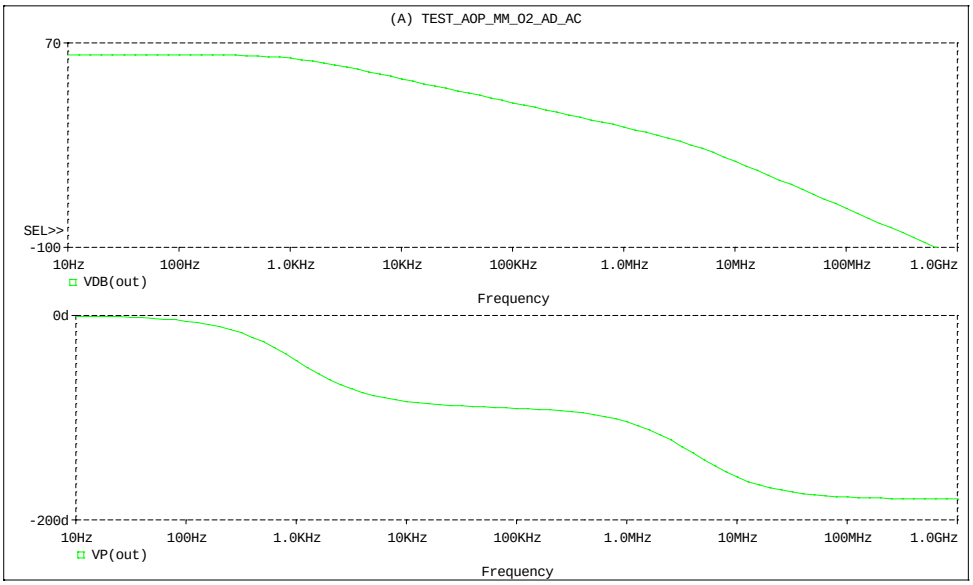


Figure 23 : Test en AC du mode différentielle.

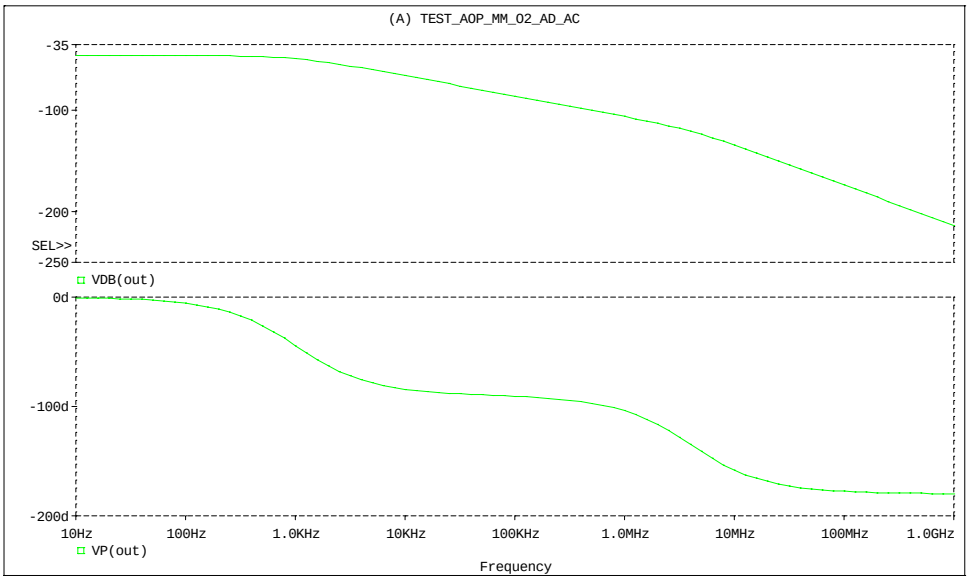


Figure 24 : Test en AC du mode commun.

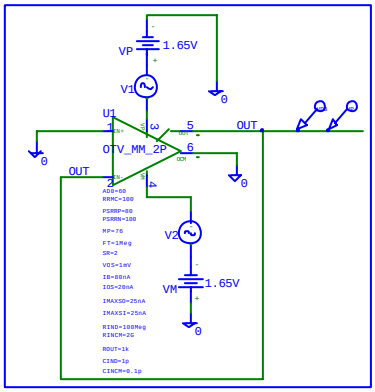


Figure 25 : Test en AC : Taux de réjection de l'alimentation (PSRR : Power Supply rejection ratio).

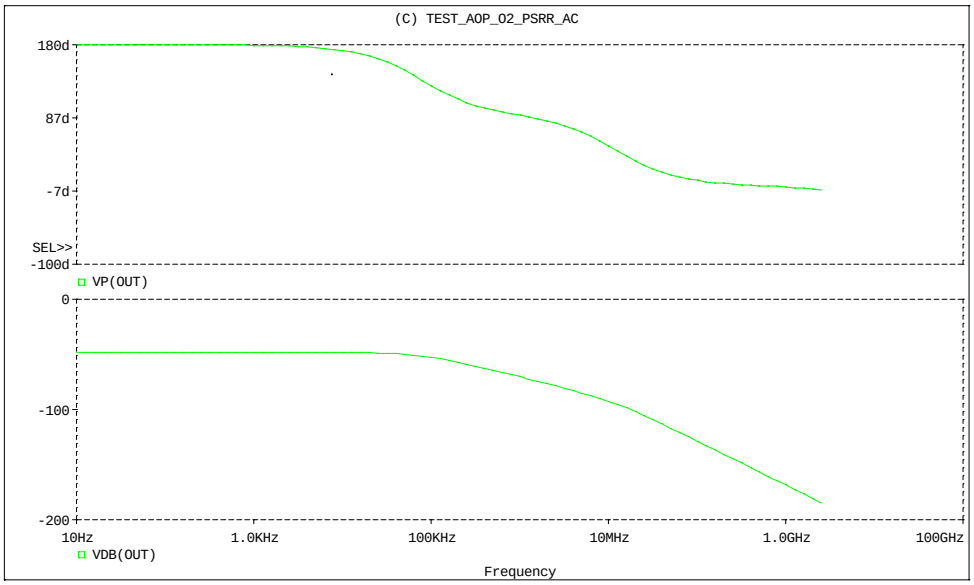


Figure 26 : Test en AC : Taux de réjection de l'alimentation.

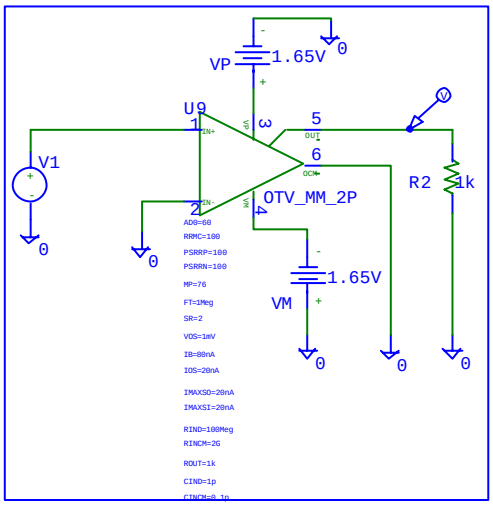


Figure 27 : Test en DC en boucle ouvert.

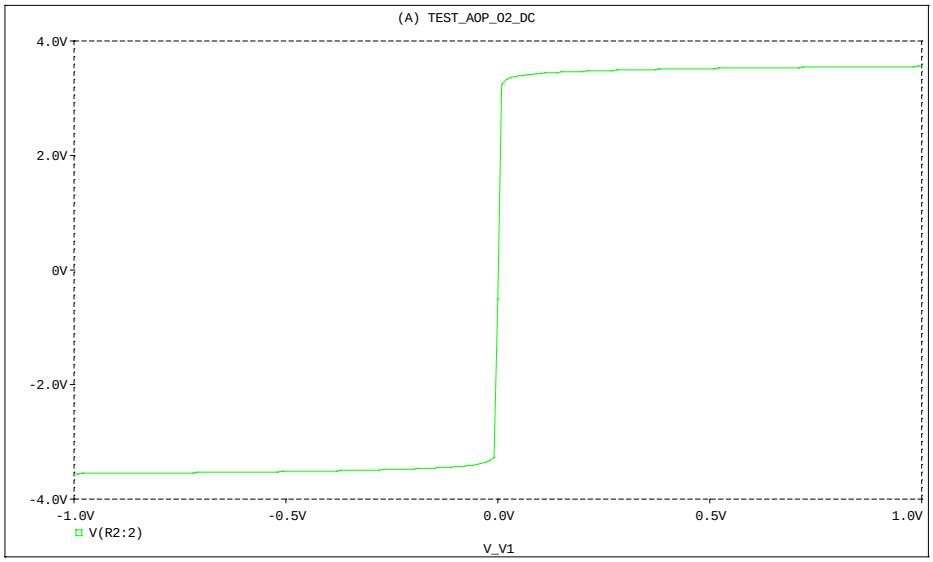


Figure 28 : Test en DC : voir la région d'amplification et la région de saturation.

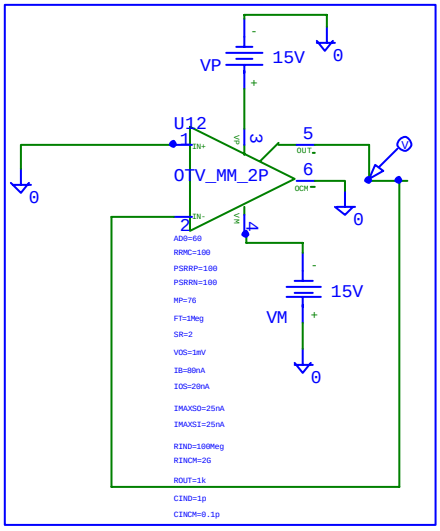


Figure 29 : Test en DC : Tension de décalage (Offset).

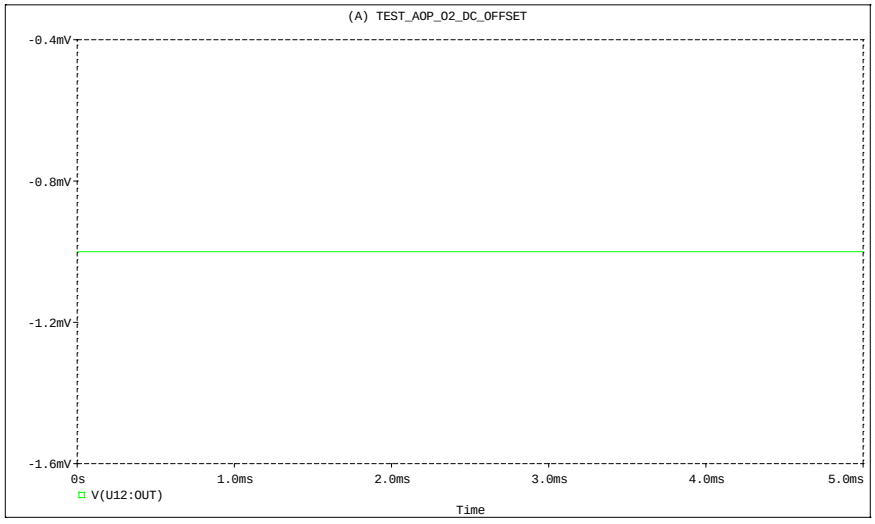


Figure 30 : Test en DC : Offset.

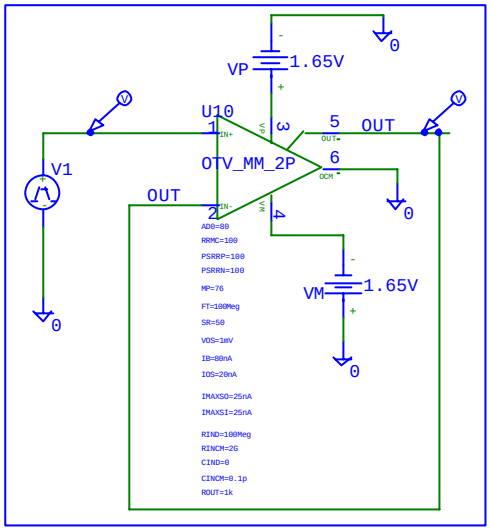


Figure 31 : Test en TRAN : Slew rate

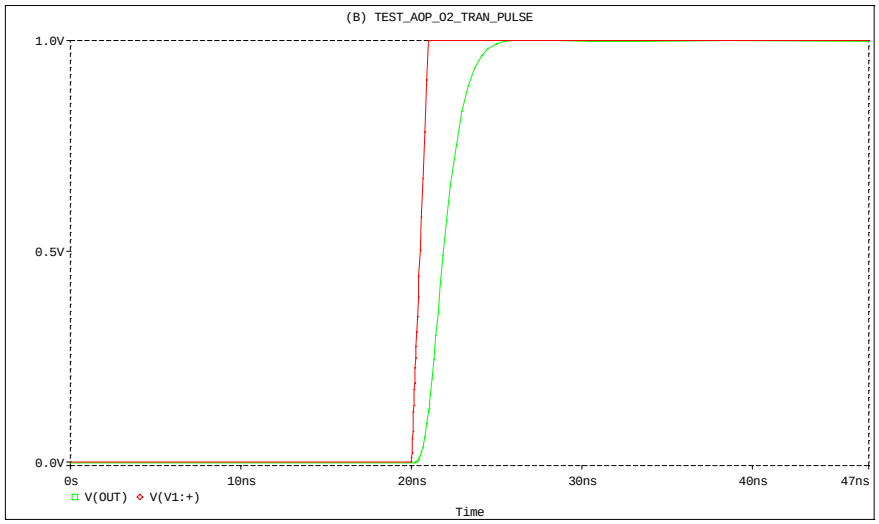


Figure 32 : Test en TRAN : Slew rate.

Annexe 5

MODELES VHDL-AMS

Exemples d'application (chapitre 2)

Modélisation fonctionnelle

Valeur absolue

```
-- VHDL-AMS automatically generated from facet mabs{ic}
ENTITY mabs IS
 GENERIC (gain :real:= 1.0001; offset :real:= 0.0);
 PORT (TERMINAL in1, in2: Electrical; QUANTITY out1: OUT Real);
END mabs;
ARCHITECTURE mabs_BODY OF mabs IS
 quantity vin across in1 to in2;
BEGIN
  out1 == gain * abs(vin) + offset;
END mabs_BOD
```

Tangente hyperbolique

```
-- VHDL-AMS automatically generated from facet tanh{ic}
ENTITY tanh IS
 --GENERIC (--Empty generic
 --        );
 PORT (TERMINAL xx, yy: Electrical);
END tanh;
ARCHITECTURE tanh_BODY OF tanh IS
  function ftanh(x1 : real) return real is
    variable y1:real;
  begin
    y1 := (exp(2.0 * x1) - 1.0) / (exp(2.0 * x1) + 1.0);
    return y1;
  end ftanh;
  quantity vxx across iyy through xx to yy;
BEGIN
  iyy == ftanh(vxx);
END tanh_BODY;
```

Différenciateur-multiplieur

```
-- VHDL-AMS automatically generated from facet difmul{ic}
ENTITY difmul IS
 GENERIC (gain :real:= 1.0001);
 PORT (TERMINAL xx, yy, gnd: Electrical; QUANTITY oo: OUT Real);
END difmul;
ARCHITECTURE difmul_BODY OF difmul IS
 quantity vxx across xx to gnd;
 quantity vyy across yy to gnd;
BEGIN
  oo == gain * (vxx - vyy);
END addmul_BODY;
```

Comparateur de tension

```
-- VHDL-AMS automatically generated from facet comparator{ic}
ENTITY comparator IS
 GENERIC (voff :real:= 0.00001; vmax :real:= 5.0001; vmin :real:=
 0.0001);
 PORT (TERMINAL in1, in2: Electrical; QUANTITY out1: OUT Real);
END comparator;
ARCHITECTURE comparator_BODY OF comparator IS
 quantity vin across in1 to in2;
BEGIN
 if (vin > voff) use
   out1 == vmax;
 else
   out2 == vmin;
 end use;
END comparator_BODY;
```

Intégrateur-gain

```
-- VHDL-AMS automatically generated from facet integral{ic}
ENTITY integral IS
 GENERIC (ic :real:= 1.001e-10;gain :real := 1.001);
 PORT (TERMINAL p, m: Electrical; QUANTITY vout: OUT Real);
END integral;
ARCHITECTURE integral_BODY OF integral IS
  quantity vin across p to m;
BEGIN
  vout == gain * vin'integ;
END integral_BODY;
```

Modélisation comportementale

Résistance

```
-- VHDL-AMS automatically generated from facet Resistance{ic}
ENTITY resistance IS
 GENERIC (r :real:= 1.00);
 PORT (TERMINAL RIN, ROUT: Electrical);
END resistance;
ARCHITECTURE resistance_BODY OF resistance IS
 quantity vr across ir through RIN to ROUT;
BEGIN
 ir == vr / r;
END resistance_BODY;
```

Capacité

```
-- VHDL-AMS automatically generated from facet Capacitance{ic}
ENTITY capacitance IS
 GENERIC (c :real:= 1.001e-012);
 PORT (TERMINAL CIN, cout: Electrical);
END capacitance;
ARCHITECTURE capacitance_BODY OF capacitance IS
 quantity vc across ic through CIN to cout;
BEGIN
 ic == c * vc'dot;
END capacitance_BODY;
```

Diode

```
ENTITY diode IS
 GENERIC (iss :real:= 1.001e-010);
 PORT (TERMINAL din, dout: Electrical);
END diode;
ARCHITECTURE diode_BODY OF diode IS
 quantity vd across id through din to dout;
BEGIN
  id == iss * (exp(vd / 26.0e-3) - 1.0);
END diode_BODY;
```

Transistor MOS correspondant au niveau 1 de modélisation SPICE

```
-- VHDL-AMS automatically generated from facet mosl1{ic}
ENTITY mosl1 IS
 GENERIC (vt0 :real:= 0.65; l :real:= 1.001e-006; gamma1 :real:= 0.58;
 lamda :real:= 0.01; w :real:= 1.001e-006; phi :real:= 1.001; kp
 :real:= 1.001);
 PORT (TERMINAL dr, gr, so, bu: Electrical);
END mosl1;
ARCHITECTURE mosl1_BODY OF mosl1 IS
  quantity vds across Ids through dr to so;
  quantity vgs across gr to so;
  quantity vbs across bu to so;
  quantity von :real;
BEGIN
von == vt0+gamma1*(sqrt(phi-vbs)-sqrt(phi));
 if (vgs<=von) use
  Ids == 0.0;
 elsif (vds<(vgs-von)) use
  Ids == kp*w/l*vds*(vgs-von-0.5*vds)*(1.0+lamda*vds);
 else
  Ids == 0.5*kp*w/l*(vgs-von)*(vgs-von)*(1.0+lamda*vds);
 end use;
END mosl1_BODY;
```

Interrupteur (switch model)

```
-- VHDL-AMS automatically generated from facet switch{ic}
ENTITY switch IS
 GENERIC (vhi :real:= 5.0001; vlo :real:= 0.0001; roff :real:= 1.0001;
 ron :real:= 1.0001);
 PORT (TERMINAL in1, out1, clk, gnd: Electrical);
END switch;
ARCHITECTURE switch_BODY OF switch IS
 quantity vclk across clk to gnd;
```

```
 quantity vron across iron through in1 to out1;
 quantity vroff across iroff through in1 to out1;
BEGIN
 if (vclk > vhi) use
    iron == vron / ron;
 else
    iroff == vroff / roff;
 end use;
END switch_BODY;
```

Convertisseur Flash

Convertisseur numérique/analogique

```
-- VHDL-AMS automatically generated from facet cda{ic}
ENTITY cda IS
 GENERIC (uref :real:= 1.0001);
 PORT (TERMINAL in0, in1, in2, gnd: Electrical; QUANTITY out1: OUT
 Real);
END cda;
ARCHITECTURE cda_BODY OF cda IS
 quantity vin0 across in0 to gnd;
 quantity vin1 across in1 to gnd;
 quantity vin2 across in2 to gnd;
BEGIN
 out1 == uref * (vin0 + vin1 * 2.0 + vin2 * 4.0) / 7.0;
END cda_BODY;
```

Comparateur

```
-- VHDL-AMS automatically generated from facet comparator{ic}
ENTITY comparator IS
 GENERIC (voff :real:= 0.00001; vmax :real:= 5.0001; vmin :real:=
 0.0001);
 PORT (TERMINAL in1, in2: Electrical; QUANTITY out1: OUT Real);
END comparator;
ARCHITECTURE comparator_BODY OF comparator IS
 quantity vin across in1 to in2;
BEGIN
 if (vin >= voff) use
   out1 == vmax;
 else
   out1 == vmin;
 end use;
END comparator_BODY;
```

Modélisation physique

Résistance

```
-- VHDL-AMS automatically generated from facet resistherm{ic}
ENTITY resistherm IS
 GENERIC (rsh :real:= 1.0e-020; res :real:= 1.0e-020; tc2 :real:= 1.0e-010;
 tc1 :real:= 1.0e-010; nrs :real:= 1.0001; m :real:= 1.0001; tnom
 :real:= 1.0e-020; rsc :real:= 1.0001);
 PORT (TERMINAL RIN, ROUT: Electrical; TERMINAL NTH, GNDT:
 Thermal);
```

```
END resistherm;
ARCHITECTURE resistherm_BODY OF resistherm IS
 constant undef : real:= 1.0e-20;
 quantity rtemp across NTH to GNDT;
 quantity vr across ir through RIN to ROUT;
 quantity delta_t, rseff, arg, rst:real;
 quantity rtnom :real;
 quantity resistance : real;
BEGIN
  IF( tnom = undef) USE
    rtnom == 27.0 + 273.15;
  ELSE
    rtnom == tnom + 273.15;
  END  USE;

  delta_t == rtemp - rtnom;
  arg == 1.0 + tc1  * delta_t + tc2  * delta_t* delta_t;
  IF(res /= undef )  USE
     IF(res = 0.0)  USE
       resistance == 0.0;
     ELSE
       resistance == res;
     END  USE;
  ELSIF (rsh /= undef)  USE
     IF(rsh /= 0.0)  USE
        IF (res > 0.0)  USE
          resistance == rsh * nrs;
        ELSE
          resistance == 0.0;
        END  USE;
     ELSE
        resistance == 0.0;
     END  USE;
  ELSE
    resistance == 0.0;
  END  USE;
  rseff == (resistance + rsc) / m;
  rst == rseff * arg;
  ir == vr / rst;
END resistherm_BODY;
```

Modélisation-simulation électro-mécanique (chapitre 5)

Modèle comportemental d'un moteur à courant continu

```
-- VHDL-AMS automatically generated from facet dcmotor{ic}
ENTITY dcmotor IS
 GENERIC (re :real:= 1.0001; krot :real:= 0.5236);
 PORT (TERMINAL in0, gnd: Electrical; QUANTITY shaft: OUT Real);
END dcmotor;
ARCHITECTURE dcmotor_BODY OF dcmotor IS
  quantity ve across ie through in0 to gnd;
BEGIN
  ie    == ve /re;
  shaft ==  krot * re * ie * ie;
END dcmotor_BODY;
```

Modèle comportemental des engrenages

```
-- VHDL-AMS automatically generated from facet gear{ic}
ENTITY gear IS
 --GENERIC (--Empty generic
 --        );
 PORT (TERMINAL in0, gndmec: Mecanical; QUANTITY out0: OUT Real);
END gear;
ARCHITECTURE gear_BODY OF gear IS
  quantity angsh across sh to gndmec;
BEGIN
  out0 == angsh'dot;
END gear_BODY;
```

Modèle comportemental du convertisseur

```
-- VHDL-AMS automatically generated from facet converter{ic}
ENTITY converter IS
 GENERIC (conv :real:= 2.0001);
 PORT (TERMINAL sh, gndmec: Mecanical; QUANTITY rotang: OUT Real);
END converter;
ARCHITECTURE converter_BODY OF converter IS
 quantity vel across angle through sh to gndmec;
BEGIN
  angle  == conv * vel;
  rotang == angle;
END converter_BODY;
```

Modèle comportemental de l'ensemble pignon crémaillère

```
-- VHDL-AMS automatically generated from facet pinionrack{ic}
ENTITY pinionrack IS
 GENERIC (tau :real:= 0.1; posmin :real:= 0.02; posmax :real:= 0.1;
 radius :real:= 0.1);
 PORT (TERMINAL pinion, gndmec: Mecanical; QUANTITY position: OUT
 Real);
END pinionrack;
ARCHITECTURE pinionrack_BODY OF pinionrack IS
 quantity apos : real;
 quantity pos1 : real;
 quantity piangle across pos through pinion to gndmec;
BEGIN
 apos == radius * piangle;
 pos1 == 0.9*posmax;
 if (apos < posmin) USE
    pos == posmin;
 else if (apos > pos1) use
         pos == 0.9 * posmax + 0.1 * posmax * (1.0 - exp((0.9 * posmax - apos) / tau));
       else
          pos == apos;
       end use;
 end use;
 position == pos;
END pinionrack_BODY;
```