Chapitre I : *Modélisation et simulation des systèmes analogiques et mixtes*

1. Notions fondamentales sur la conception des systèmes

Du fait de la complexité et de l'hétérogénéité des systèmes électroniques, les concepteurs doivent gérer des projets associant plusieurs disciplines et plusieurs technologies.

1.1 Description de la conception hiérarchique

Récemment, en raison de l'hétérogénéité et de la grande complexité des systèmes intégrés sur une puce, la hiérarchisation de la conception s'avère nécessaire. Autrement dit, le concepteur commence par concevoir et valider un système à l'aide de blocs fonctionnels, puis il descend progressivement dans le détail des blocs, jusqu'à la conception de circuits élémentaires au niveau transistor ou portes logiques.

Lors de la conception d'un système, le problème initial étant la traduction du cahier des charges en un circuit intégré fonctionnel. On distingue deux modes de conception d'un système correspondant aux deux sens de parcours de l'hiérarchie : *la conception en mode descendant (Top-Down)* et *la conception en mode ascendant* (*Bottom-Up*). Chaque niveau hiérarchique est caractérisé par un ensemble d'entités permettant de décrire la topologie du système.

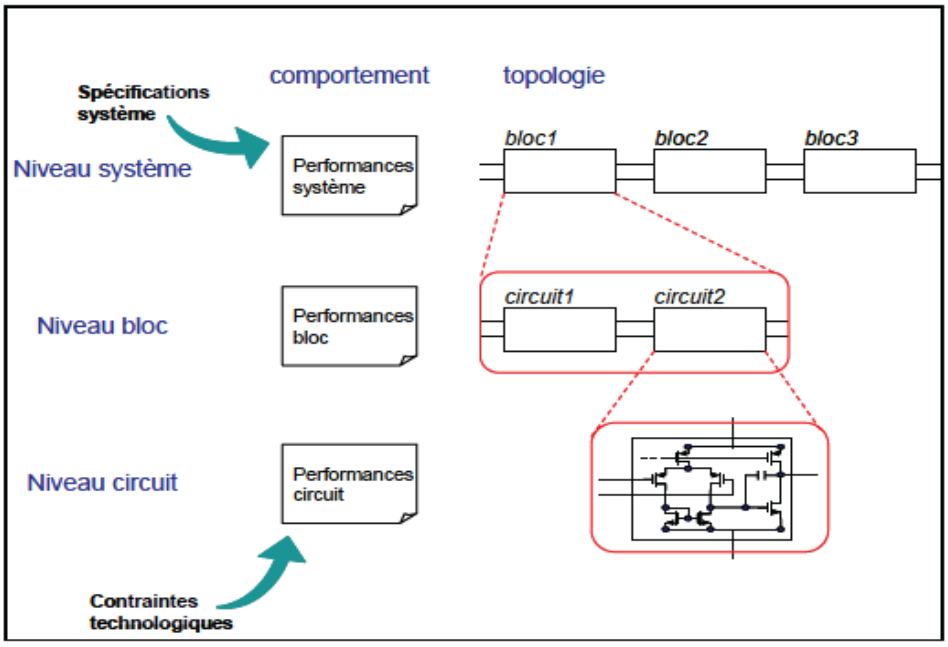

***Figure I.1** : Description de la conception hiérarchique.*

La figure I.1 présente une description générale de la conception hiérarchique en mode descendant. Dans ce cas, le système est décomposé en trois niveaux hiérarchiques :

- **Niveau système :** la topologie du système est décrite au moyen de blocs fonctionnels.
- **Niveau bloc :** les primitives de chaque bloc sont des circuits.
- **Niveau circuit :** les primitives à ce niveau sont des composants électroniques de base (transistors, diodes, résistances,…).

La conception hiérarchique est contrainte au plus haut niveau par les *spécifications système* ainsi qu'au plus bas niveau par les *contraintes technologiques.* Les spécifications système sont les données du client fixant les performances du système, tandis que les contraintes technologiques imposent les marges de conception.

1.2 Modèle et modélisation

- Le modèle consiste essentiellement à développer une représentation abstraite d'une réalité physique. Le modèle dépend du point de vue selon lequel on observe le système, mais aussi suivant l'utilisation que l'on souhaite faire de ce modèle au sein du processus de conception.
- La modélisation représente la tâche centrale de la conception. Elle consiste à trouver une loi mathématique représentative du comportement d'un système et la vérification de la vraisemblance de cette loi se fait par comparaison avec des données de référence provenant des mesures avant tous et parfois des simulations réalisées à partir des modèles déjà validés.

2. Méthodologies de modélisation

La modélisation comportementale signifie le processus partant de l'analyse du comportement d'un circuit et aboutissant à un système d'équations. Ce processus diffère selon les phases de la conception d'un système : en phase descendante ou en phase montante.

2.1 Modélisation en phase Bottom-Up (ascendante)

Dans la phase de conception ascendante (Bottom-Up), le point de départ est un circuit dimensionné ayant un objectif précis. Il s'agit d'en extraire un modèle comportemental, nécessairement plus abstrait que la description au niveau transistor, mais capable de propager les performances du circuit réalisé vers les haut niveaux hiérarchiques.

2.2 Modélisation en phase Top-Down (descendante)

Dans cette phase de conception, le schéma électrique étant inconnu et le point de départ est une liste de spécifications. L'objectif est alors de construire un modèle décrivant la fonction du bloc considéré ajustable par des paramètres de performances.

Les modèles extraits en phase Top-Down sont des modèles *fonctionnels* caractérisés par une précision moindre que celle des modèles *comportementaux* extraits en phase Bottom-Up. Tandis que la rapidité est prédominante dans cette phase.

3. Simulation des systèmes mixtes (analogique-digitale)

La simulation mixte permet d'étudier le comportement temporel de systèmes complexes en un temps extrêmement réduit par rapport à une simulation uniquement électrique. Ce type de simulation est en effet basé sur l'abstraction de la partie digitale à un niveau fonctionnel logique. Pour cette partie, les grandeurs étudiées ne sont donc plus électriques mais numériques et sont caractérisées par leurs changements d'état. Une simulation s'effectue en trois différentes phases:

3.1 Phase d'élaboration

Elle correspond à la décomposition du circuit mixte en blocs distincts analogiques et digitaux, chaque partie étant traitée par les algorithmes concernés. Aux interfaces entre les deux parties, doivent être placés des modèles plus ou moins élaborés de *convertisseurs* A/D et D/A, qui assurent la correspondance des données entre les algorithmes analogiques et digitaux.

3.2 Phase d'initialisation

Il s'agit de déterminer le point de fonctionnement du système, c'est à dire l'état initial de toutes les grandeurs mises en jeu (tensions, courants, états logiques). Cette recherche est indispensable au simulateur analogique et correspond à une analyse DC. Pour la partie digitale, cette notion dépend du simulateur: cela peut correspondre soit à une initialisation (solution au temps 0), soit à une certaine durée, appelée temps de *setup*, au bout de laquelle un état stable est trouvé.

3.3 Phase de simulation

Elle doit résoudre les problèmes de *synchronisation* des algorithmes électriques et digitaux, qui ont des gestions différentes du pas de temps.

Chapitre II : *Simulation analogiques des circuits électroniques*

1. Notions sur la simulation analogique

- Un simulateur électrique est un programme informatique, qui à partir de la description d'un circuit et de ses variables d'excitation permet de calculer n'importe qu'elle caractéristique ou variable électrique (tension, courant, impédance,…) en n'importe quel endroit d'un circuit et quelles que soient les excitations appliquées.
- Le logiciel SPICE est la référence en matière de simulateur analogique des circuits intégrés. Il existe de nombreuses versions industrielles basées sur le même langage de description structurelle comme PSPICE, LTSPICE, ORCAD-SPICE,...etc.
- Une bibliothèque de composants qui sont modélisés dans le code même du logiciel de simulation est fournie comportant des éléments passifs (résistances, capacités, inductances, inductances mutuelles), des composants semi-conducteurs (diodes, transistors bipolaires, à effet de champ JFET et MOSFET), des sources idéales indépendantes de tension et de courant et enfin des sources idéales contrôlées.
- Grâce à un logiciel de simulation électrique, on peut vérifier la conformité des résultats expérimentaux sans passer par la phase d'élaboration du prototype, surtout pour les circuits de grande complexité comportant un grand nombre d'éléments non linéaires.

2. Procédure interne d'un simulateur électrique

2.1 Principe de base d'un simulateur

En général, les langages de simulation analogique se servent de modèles d'équations représentant le comportement physique de différents composants. La précision de ces modèles dépend de la complexité des équations et aussi des nombre de paramètres.

Les simulateurs analogiques sont consacrés à analyser le contenu fréquentiel et temporel des circuits. De ce fait, ils disposent des algorithmes de résolution numériques des équations différentielles.

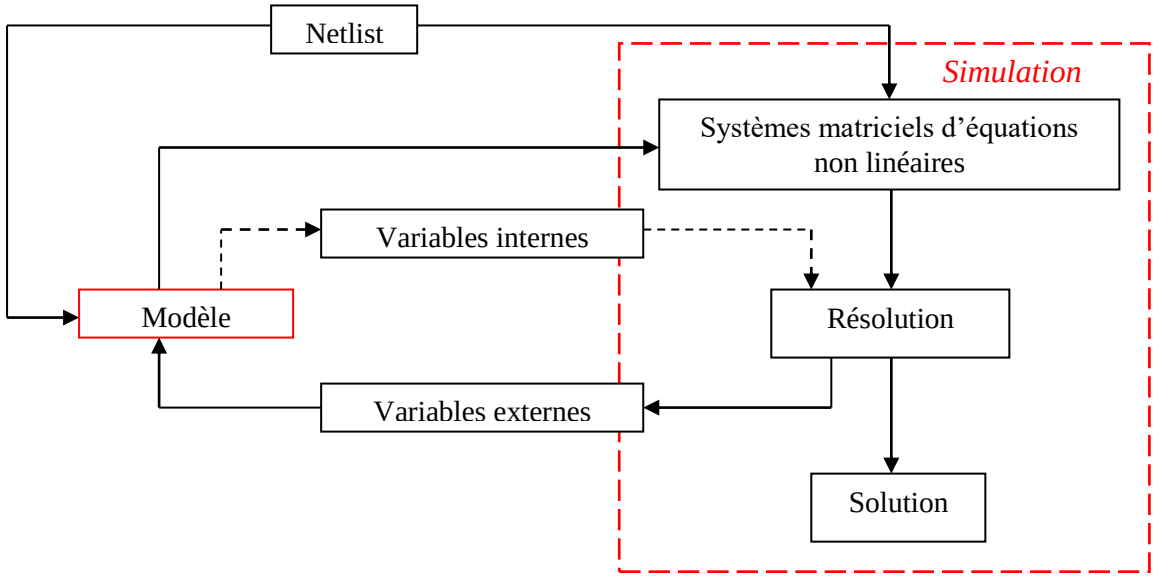

***Figure II.1**: Principe général d'un simulateur électrique.*

La figure II.1 montre que les circuits sont décrit par une liste des interconnexions, appelée *Netlist*, indiquant comment sont connectés le composants. A chaque modèle, un système d'équations décrivant les lois aux différents nœuds (tension et courant) est associé. Le simulateur résout ces systèmes d'équations non linéaires par des méthodes d'intégration numérique, des techniques itératives et des méthodes de résolution matricielles.

2.2 La mise en équation des circuits électriques

Un circuit électrique constitué d'un ensemble de branches, peut être décrit par un système d'équations satisfaisant aux équations de Kirchhoff des courants et des tensions, ainsi qu'aux équations des composants. Chaque composant, considéré comme une interconnexion de branches, exprime les relations des tensions ou courants de ses branches, en fonction des inconnues du circuit.

A partir de la figure II.2, le modèle du composant amplificateur composé par deux branches *AB* et *CD* et par conséquent se caractérise par deux relations définissant les courants qui y circulent.

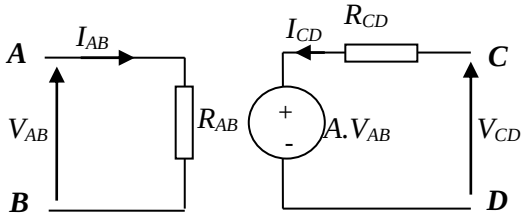

***Figure II.2**: Modèle d'un amplificateur linéaire.*

Pour exprimer le système d'équations correspondant à ce réseau électrique, l'algorithme de l'*analyse nodale* est souvent utilisé. En effet, cette méthode conduit à un système d'équations très simple, dont les variables indépendantes du système sont les tensions des nœuds, calculées par rapport à un nœud de référence (la masse). Dans ce cas, le système d'équations est composé des équations de Kirchhoff des courants en chaque nœud, à l'exception du nœud de référence. Le système d'équations est donc exprimé comme suit :

$$\begin{cases} I_{AB} = \frac{V_{AB}}{R_{AB}} \\ I_{CD} = \frac{V_{CD} - A.V_{AB}}{R_{CD}} \end{cases} \quad (1)$$

Cette méthode présente cependant une limitation dans le cas d'utilisation de sources de tension dont le courant qui les traverse est une inconnue. Pour remédier ce problème, l'utilisation des méthodes hybrides, telles que l'*analyse nodale modifiée* (Modified Nodal Analysis ou MNA) s'est avéré nécessaire: cette méthode, utilisée dans SPICE, offre un bon compromis simplicité/généralité. Les variables indépendantes du système sont constituées des tensions des nœuds et des courants circulant dans les sources de tension et dans tous les composants dont les équations de branche ne peuvent être représentées sous la forme explicite :

$$i = Y.v \quad (2)$$

où Y *:* représente l'admittance de la branche.

Les équations du système sont donc constituées des équations de Kirchhoff des courants et des équations de branches pour lesquelles le courant est une inconnue. Enfin, quelle que soit la méthode utilisée, la taille du système à résoudre dépend directement du nombre de nœuds du réseau électrique.

.

Chapitre III : *Langages de description des systèmes mixtes*

1. Simulation des signaux analogiques-numériques

Le logiciel PSPICE représente un simulateur complet pour la conception analogique. Du fait de ses modèles internes et ses bibliothèques largement rependues et développées, tous les systèmes de haute fréquence jusqu'aux circuits intégrés de basse puissance peuvent être simulés. Dans sa bibliothèque, des modèles peuvent être édités et/ou des modèles de nouveaux dispositifs peuvent être créés à partir des fiches techniques.

La version plus élaborée de PSPICE dite « PSPICE A/D Basics » est un simulateur de signaux mixtes pouvant être employée pour simuler des systèmes contenant des parties analogiques et des éléments numériques sans limite théorique de la taille. Cependant, quand il s'agit de grands systèmes, les simulations deviennent trop lourdes et demandent un temps d'exécution exagéré.

Par conséquent, les exigences de la technologie et du marché ont imposé le développement d'outils plus vigoureux capables de traiter simultanément les domaines analogiques et numériques. Ce besoin a entraîné depuis la fin des années 90, l'apparition de langages de description matérielle de systèmes à signaux mixtes MSHDLs (*Mixed Signals Hardware Description Langages*). Ces types de langages offrent un grand intérêt dans une approche de conception système.

2. Langages de modélisation numériques

Les langages de modélisation numérique sont les langages de description matérielle de haut niveau communément appelé HDL (*Hardware Description Language*). Ce dernier représente une instance d'une classe de langage informatique ayant pour objectif la description formelle d'un système électronique.

Parmi les fonctions pouvant être réalisées par le langage HDL, on peut citer :

- décrire le fonctionnement et la structure du circuit,
- assurer sa documentation,
- permettre des preuves formelles,
- aider à co-vérifier de netlist ,
- tester le circuit en le vérifiant par simulation.

Il existe plusieurs types de langages de description de matériel (HDL), dont les plus connus sont :

3.1 Langage Verilog

Le langage Verilog était à l'origine un langage propriétaire (non libre) de description de matériel, développé par la société "Cadence Design Systems", pour être utilisé dans leurs simulateurs logiques. Ce langage permet d'une part de décrire l'enchainement d'événements et d'autre part de synthétiser des circuits numériques par combinaison de plusieurs éléments logiques (modules, portes logiques,…). La syntaxe de « Verilog » est largement inspirée du langage de programmation C.

3.2 Langage VHDL

Le langage VHDL (*Very High Speed integrated circuits Hardware Description Langage*) est un langage lisible, actuel et puissant de description de matériel destiné à décrire le comportement et/ou l'architecture d'un système électronique numérique.

L'intérêt d'une telle description réside dans son caractère exécutable, c.-à-d. une spécification fonctionnelle décrite en VHDL peut être vérifiée par simulation avant la finalisation de la conception détaillée du système. La syntaxe du VHDL est originaire du langage ADA.

Exemple: Décodeur 2 vers 4

Nous considérons un circuit combinatoire qui est un décodeur 2/4 (2 entrées et 4 sorties) représenté par la figure III.1. Il est réalisé avec des portes « inverseur » et des portes « AND ». Dans ce cas la modélisation numérique par le langage VHDL est illustrée sur l'encadré III.1.

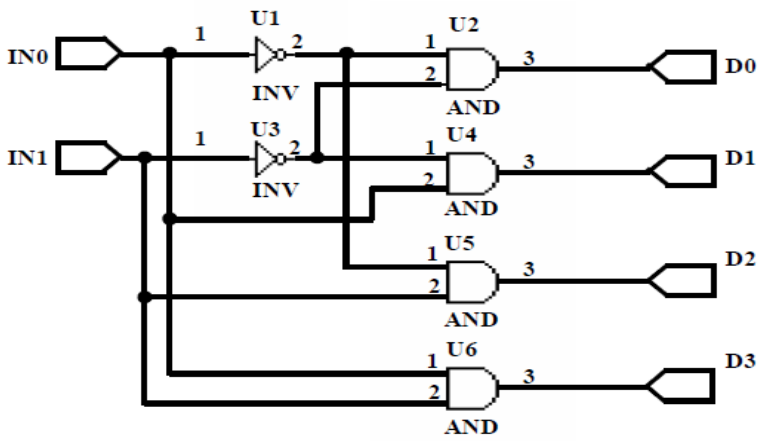

Figure III.1: *Décodeur 2/4.*

IN0	IN1	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	0	0	1

Tableau III.1: *Table de vérité du décodeur 2/4.*

```
entity décodeur is
--Définition des entrés sorties
…
end entity décodeur ;
architecture description of décodeur is
begin
IN0 <= not(IN0) after1.0 ms;
IN1 <= not(IN1) after2.0 ms;
D0 <= not(IN0) and not(IN1);
D1 <= IN0 and not(IN1);
D2 <= not(IN0) and IN1;
D3 <= IN0 and IN1;
end description;
```

Encadré III.1: *Code VHDL du décodeur 2/4.*

3. Langages de modélisation mixte multi-domaine

Un modèle mixte inclut des parties ayant un comportement continu (parties analogiques) et des parties ayant un comportement dirigé par événements (parties logiques) qui interfèrent. À l'aide des langages de modélisation mixte de haut niveau, les différentes phases de conception d'un système peuvent être optimisées. Ces langages permettent de traiter indifféremment des modélisations logiques, analogiques ou mixtes au sein d'un même composant ou système.

Parmi ces langages, ont peut citer : VHDL-AMS, Verilog-AMS, MAST, Modelica et la notation Bond Graph.

4.1 Verilog AMS

Verilog-AMS est un langage de description du matériel (HDL) permettant d'exprimer de modèles et de systèmes à temps continu et à temps discret. Il a été créé sous la tutelle d'Accellera (Organisation de standards EDA : *Electronic Design Automation*) afin de mettre en place les extensions analogiques mixtes de Verilog. La version la plus récente est « Verilog-AMS LRM-2.2 » sortie en novembre 2004.

Ce langage permet de faire la description comportementale des systèmes analogiques et mixtes. Ainsi, ce langage peut être applicable aux systèmes électriques et non électriques. Des descriptions de systèmes peuvent être effectuées sous ce langage, en utilisant des concepts comme des nœuds, des branches et des ports.

4.2 VHDL-AMS

Le langage VHDL-AMS inclut toutes les propriétés du VHDL standard, avec en plus, l'aptitude de décrire les systèmes mixtes à travers de modèles multi abstractions, multidisciplinaires et/ou hiérarchiques à temps continu et à événements discretsgure III.2). La dernière version publiée est « IEEE Std 1076.1 2007, sortie en 15 novembre 2007.

L'avantage de ce langage non propriétaire est de proposer un langage commun indépendant des fournisseurs et de la technologie. Du point de vue technique, il permet une haute modularité facilitant des descriptions hiérarchiques. Tous les systèmes qui peuvent être modélisés sous Matlab, VHDL et Spice peuvent aujourd'hui être modélisés sous un seul langage (VHDL-AMS). En effet, ce dernier peut travailler simultanément comme un simulateur à évènements discrets et un solveur d'équations différentielles.

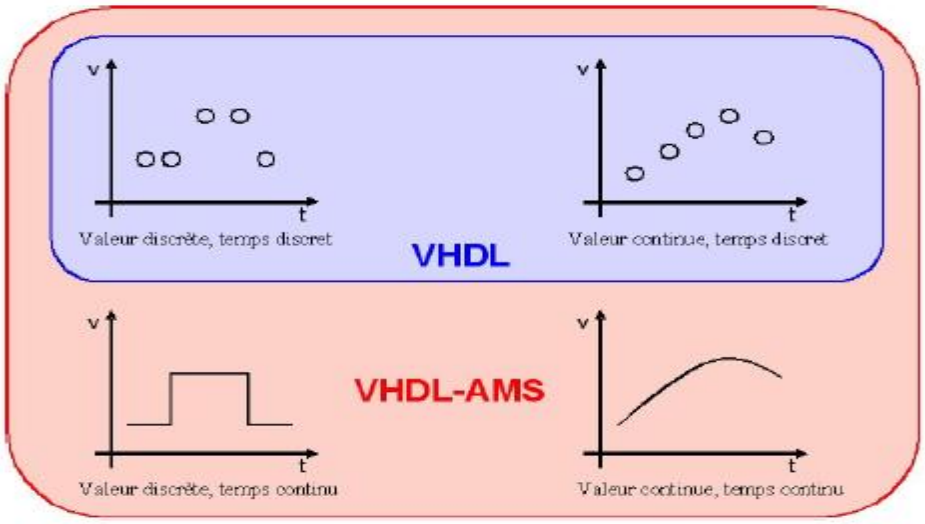

Figure III.2 : Couverture du langage VHDL-AMS.

Chapitre IV : *Conception des Circuits électroniques analogiques-numériques via VHDL-AMS*

1. Environnement de travail du langage VHDL-AMS

La figure IV.1 représente l'environnement de travail du standard VHDL-AMS contenant différentes phases d'édition, d'analyse, d'élaboration et d'exécution :

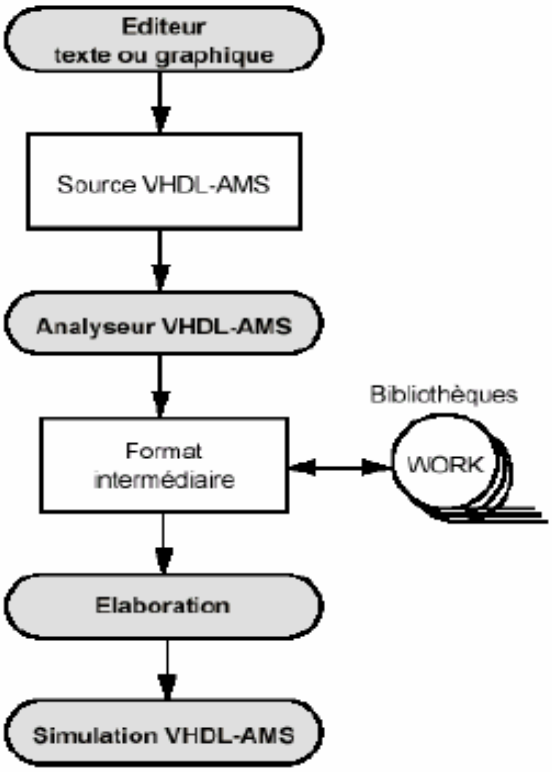

Figure IV.1**:** *Environnement travail VHDL-AMS.*

1.1 L'interface graphique : elle peut se réduire à un simple éditeur de texte. Les outils CAO du marché utilisent en plus leur éditeur de schémas pour générer automatiquement le squelette d'un modèle VHDL-AMS. Des outils plus avancés permettent de décrire le comportement du système à modéliser sous la forme de machines d'états, de chronogrammes ou de tables de vérité.

1.2 L'analyseur (compilateur) : son rôle du compilateur est la vérification de la syntaxe d'une description VHDL-AMS. Il permet la détection d'erreurs locales, qui ne

concernent que de l'unité compilée. Plusieurs techniques d'analyse sont actuellement utilisées par les outils du marché.

1.3 Bibliothèque de travail (Working library) : chaque concepteur possède une bibliothèque de travail de nom logique WORK (le nom est standard) dans laquelle sont placés tous les modèles compilés.

1.4 Le simulateur : il calcule comment le système modélisé se comporte lorsqu'on lui applique un ensemble de stimuli. L'environnement de test peut également être écrit en VHDL-AMS: il peut être lui-même vu comme un système définissant les excitations et les opérations à appliquer aux signaux de sortie pour les visualiser (sous forme texte ou graphique).

1.5 La phase d'élaboration

Elle consiste en une construction des structures de données et permet la détection d'erreurs globales, qui concernent l'ensemble des unités de la description. Cette phase est normalement exécutée en arrière-plan avant la simulation proprement dite.

2. Description structurelle d'un modèle VHDL-AMS

2.1 Structure générale d'une entité de conception VHDL-AMS

La modélisation en VHDL-AMS de tout composant s'effectue au moyen de deux types d'objets : entity et architecture constituant tous les deux une entité de conception VHDL-AMS.

a. L'entité (entity)

La déclaration d'entité définit l'interface d'un modèle avec le monde extérieur au moyen des ports. On peut comparer l'entité a une boite noire où seules les entrés-sorties du composant sont visibles. Lors de l'écriture d'une entité, on ne déclare que l'interface avec le monde extérieur grâce aux mots **port** et **generic**.

generic: regroupe la déclaration des paramètres du composant.

port: décrit l'interface avec l'environnement extérieur par l'intermédiaire des nœuds externes.

Exemple de syntaxe :

entity nom_de_l'entité **is**
generic (generic_déclarations);
port (port_déclarations);
end entity nom_de_l'entité

b. L'architecture

Une architecture définit le comportement et/ou la structure du système modélisé. Une architecture se réfère toujours à une entité unique et contient la description de la fonction réalisée par cette dernière. Pour une entité donnée réalisant une fonction précise, il peut y avoir autant d'architectures de manières de décrire la fonction à réaliser.

Exemple de syntaxe :

architecture nom_de_l'architecture **of** nom_de_l'entité **is**

déclaration_des_variables

begin

déclaration_des_équations

end architecture nom_de_l'architecture

2.2 Déclaration des quantités et des terminaux

Une vue interne (architecture) possible en VHDL-AMS est une description structurelle pour laquelle le modèle est une interconnexion de composants, avec éventuellement un nombre de niveaux hiérarchiques non limité.

a. Quantité (quantity)

Une quantité sert à modéliser une quantité physique électrique, mécanique thermique…etc. Contrairement aux signaux et aux variables qui ne changent de valeur qu'aux instants précis appelés événements les quantités sont des fonctions continues du temps (ou de la fréquence). Les quantités peuvent être :

- Des quantités libres : qui sont déclarées à l'intérieur d'une architecture :

  quantity omega : real ;

- Des quantités sources : Les quantités sources permettent de définir les signaux utilisés pour les analyses en fréquence AC et NOISE.

  quantity iac : **real spectrum** 1.0 , 0.0 ; --Magnitude Phase

  quantity inn : **real noise** 4*k*T/R ;

  i == V/R + iac + inn; -- source de courant de résistance interne R

- Des quantités d'interface : Les quantités d'interface permettent de faire de la modélisation de type schéma-bloc (signal-flow). Les quantités d'interface sont déclarées dans la déclaration de port d'une entité. Les quantités sont de mode in ou out.

entity ampli **is**

port(**quantity** Vip, Vin : **in** real);

quantity Vout : **out** real;

signal Enable : **in** bit);

end entity;

- Des quantités de branche : elles sont associées aux terminaux :
  - Les quantités across représentent un effort : différence de potentiel électrique, différence de température, différence de pression ….
  - Les quantités through représentent un flux : courant électrique, puissance themique, débit volumique…

b. Terminaux (terminal)

Un terminal correspond à une équipotentielle d'un système physique conservatif décrit par un graphe. Le domaine physique auquel appartient le terminal est la nature du terminal la déclaration d'une nature définit le type des quantités across et through ainsi que la reference.

Exemple : Les déclarations ci- dessous sont correspondantes au circuit présenté sur la figure IV.2.

library disciplines;

use disciplines.electromagnetic_system.all;

terminal T1, T2, T3, T4 : electrical;

quantity V1 **across** I1 **through** T1 to T2;

quantity V2 **across** I2 **through** T3; -- le deuxième terminal est la référence

quantity V3 **across** I3 **through** T3; -- V3 est identique à V2

quantity V4 **across** T3 to T4; -- ne crée pas de branche

quantity I4 **through** T4 ;

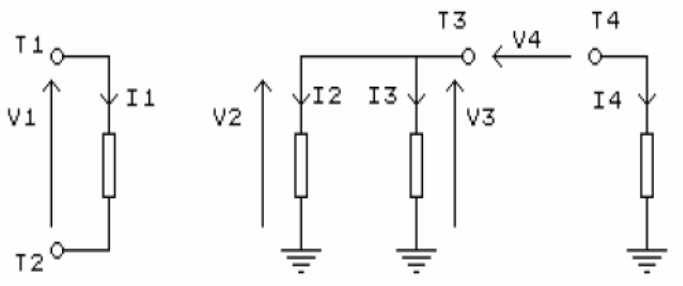

Figure IV.2: *Exemple d'un circuit électrique.*

3. Exemple de modélisation en VHDL-AMS des circuits électriques

3.1 Modélisation des composants élémentaires

- **Résistance :**

Le courant traversant une résistance est défini selon la loi d'Ohm par:

$$i_R(t) = \frac{u_R(t)}{R} \quad (1)$$

Syntaxe :

entity resistor **is**

generic (résistance : real := 1.0); -- déclaration d'une valeur par défaut de la résistance.

port (**terminal** n1, n2 : electrical);

end resistor

architecture behav **of** resistor **is**

quantity r_e across r_i through n1 to n2;

begin

r_i == r_e/résistance; -- Calcul classique du courant à travers une résistance.

end behav;

- **Capacité :**

Le courant traversant un condensateur est décrit par l'expression suivante :

$$i_C(t) = C.\frac{du_C(t)}{dt} \quad (2)$$

Syntaxe :

entity capacitor **is**

generic (capacité : real := 10.0e-9); -- déclarer une valeur par défaut de capacité

port (**terminal** n1, n2 : electrical);

end capacitor;

architecture behav **of** capacitor **is**

quantity c_e across c_i through n1 to n2;

begin

c_i == capacité*c_e'**dot**;

end behav;

- **Inductance :**

Le courant traversant une inductance est exprimé par l'équation :

$$i_L(t) = \frac{1}{L}.\int u_L(t)dt \qquad (3)$$

Syntaxe :

Entity inductor **is**

generic (L : real := 1.0); -- valeur par défaut obligatoire.

port (**terminal** n1, n2 : electrical);

end inductor;

architecture behav **of** inductor **is**
quantitty L_e across L_i through n1 to n2;
begin
L_i == (L_e'**Integ**)/L; -- utilisation de la quantité implicite Q'Integ
end behav;

3.2 Modélisation d'un circuit RLC en VHDL-AMS

A titre d'exemple, considérons le circuit RLC représenté sur la figure IV.3. La modélisation sous VHDL-AMS de ce circuit consiste tous d'abord à définir chaque composant séparément (source de tension, résistance, inductance et capacité), puis la définition du circuit global comme il est illustrée ci-dessous :

entity circuit **is**
end;
architecture behavioral **of** circuit **is**
terminal n1,n2,n3: ELECTRICAL;
begin
vsrc: entity voltg (behavioral) port map (n1, electrical_ground);
r1: entity resistor (behavioral) **port** map (n1, n2);
l1: entity inductor (behavioral) **port** map (n2, n3);
c1: entity capacitor (behavioral) **port** map (n3, electrical_ground);
end;

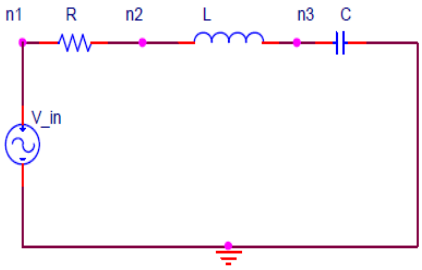

Figure IV.3: *Circuit RLC série.*