Chapitre II

VHDL-AMS et méthodologie de simulation des systèmes multidisciplinaires

Table de matière du chapitre II

II.1 Introduction

La modélisation et la simulation sont des tâches primordiales sur le chemin de la conception des systèmes multidisciplinaires. La simulation, avant d'entamer les démarches de matérialisation et de réalisation technologique, permet de corriger plus facilement les erreurs éventuelles et d'optimiser le coût de développement et d'industrialisation. Par ailleurs, elle permet aussi d'envisager des scénarios non mesurables sur des composants réels sans les détruire.

Les progrès faits ces dernières années sur la conception des systèmes de transmission à fibre optique permettent l'utilisation des composants optoélectroniques pour des équipements de plus en plus puissants, demandant un effort préalable de modélisation plus important. La complexité croissante des circuits et des systèmes optique actuels nous pousse à considérer des systèmes multidisciplinaires, mettant en œuvre à la fois l'électronique numérique, l'électronique analogique, la thermique, l'optique, la mécanique d'autres disciplines encore.

Ces enjeux pluridisciplinaires et le besoin d'optimiser le processus de conception pour réduire le « time to market » conduisent au développement de nouvelles techniques telles que la modélisation et la validation à haut-niveau, la modélisation fonctionnelle, la réutilisation et la génération de modules de propriété intellectuelle (IP) **[II.1]**, …etc. Ces nouvelles approches doivent être considérées dès les premières étapes de la conception. Actuellement, le concepteur du système multidisciplinaire doit répondre aux contraintes industrielles fortes :

- Modélisation fonctionnelle, exécutables et échangeables pour vérifier par simulation tout au long du processus de conception, la conformité au cahier des charge, mais aussi pour optimiser les performances du système,
- Il faut aussi assurer la formalisation et la capitalisation des modèles développés **[II.2]**, avec lesquels nous sommes capables de stocker des modèles dans des bibliothèques de composants généralistes et bien documentés autorisant la réutilisation intensive de ces derniers, afin d'économiser le temps de développement de nouveau système. De nombreux projets ont été mis en place afin d'améliorer la réutilisation des modèles comme la plateforme DIMOCODE **[II.3]**.
- Gestion des équipes de conception des systèmes complexes afin de mieux résoudre les questions de collaboration.

Ainsi, afin de garantir un niveau de confiance élevé dans le flot de conception tout en assurant une durée de développement limitée et un coût réduit, nous avons besoin de

modéliser et simuler le système de transmission complet dans un environnement de simulation adapté et unique, en prenant en compte leur complexité et leur pluridisciplinarité.

Le langage normalisé VHDL-AMS « IEEE Std 1076.1 2007 » est le plus adapté pour répondre à nos besoins. Puisqu'il faut assurer la capitalisation et la réutilisation des modèles développés pour augmenter la productivité du travail de conception, nous développerons une bibliothèque de modèles des composants optoélectronique. Cette bibliothèque sert à documenter des modèles de composants optoélectronique qui sont essentiels à la modélisation des systèmes qui les utilisent.

II.2 Choix du langage de modélisation

Après la détermination des niveaux de modélisation nécessaires pour les différents blocs constituant notre système, il nous a fallu trouver le langage de modélisation le mieux adapté à nos besoins. Les langages susceptibles d'apporter une réponse aux problèmes de modélisation sont nombreux, mais répondent peu à nos attentes.

II.2.1 Langage de modélisation numérique

Les langages de modélisation numérique sont les langages de description matérielle de haut niveau communément appelé HDL. Un langage HDL est une instance d'une classe de langage informatique ayant pour but la description formelle d'un système électronique. Le HDL « numériques » ou « orienté synthèse » est un langage de description de circuits logiques en électronique, utilisé pour la conception d'ASICs et FPGAs **[II.4]**.

HDL peut généralement :

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

Comme exemple nous présentons le langage « Verilog » qui 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. L'Institut d'Electrique et d'Electroniques des Ingénieurs (IEEE) a normalisé le langage « Verilog » comme un langage de description pour les modèles numériques, cette norme est définie par le standard : IEEE

Std 1364-2005 **[II.5]**. La syntaxe de « Verilog » est réputée largement inspirée du langage de programmation "C" **[II.6]**.

II.2.2 Langage de modélisation analogique

II.2.2.1 VHDL

Dans les années quatre-vingt, le département de la défense américaine fait un appel d'offre afin de disposer d'un langage de description de matériel numérique unique. Le langage VHDL est retenu pour répondre à ce critère. Ce langage est un sous-produit du programme VHSIC qui est un projet de recherche national mené par le groupement IBM / Texas Instruments / Intermetrics. Ce langage est ouvert au domaine public en 1985 et deviendra une norme en 1987 sous la dénomination de IEEE 1076-1987. Révisée en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques. Il est l'outil qui a ouvert la voie de la conception et de la synthèse automatique de circuits logiques sur une base normalisée. La norme en cours est connue sous le nom de VHDL 4.0.

En parallèle à la révision de 1993 du VHDL, un sous-programme par 1071.1 avait pour objectif d'étende le langage aux systèmes analogiques. La révision IEEE 1076.1-2007 **[II.7]** inclut les modifications depuis la première version de la norme IEEE 1076.1 standard. Elle comprend également la clarification des définitions IEEE 1076.1et des corrections d'erreurs typographiques du manuel de référence du langage.

Le VHDL est un langage de haut niveau. Il permet de décrire d'un point de vue purement fonctionnel, le comportement des systèmes numériques. Il permet également de synthétiser des circuits numériques, c'est-à-dire de traduire en schéma logique (netlist de ressources de base) leur description comportementale **[II.8],** de programmer des composants programmables de type FPGA, et de tester le composant dans un système complexe avant même de l'acheter ou d'attendre qu'une autre équipe ne l'ai développé. L'une des particularités du VHDL provient du fait qu'il est possible d'exprimer facilement le parallélisme à l'intérieur d'un circuit **[II.9]**.

Au début le VHDL était limité au seul domaine de l'électronique numérique et faisait défaut lors de la description des circuits électroniques mixtes. Depuis plusieurs années les industriels et les sociétés de conception de logiciel d'ingénierie électronique proposent des solutions en créant leur propre langage. Là encore l'IEEE après un effort de standardisation

conduira en décembre 1999 à la normalisation de l'extension du VHDL pour aboutir au VHDL 1076.1 aussi appelé VHDL-AMS (VHDL-Analog Mixed Signal).

II.2.2.2 Types de description utilisables en VHDL

La conception d'un système passe par sa description toujours réalisée en deux étapes au minimum. La première étape consiste à décrire le système comme une boîte noire, alors que la seconde s'intéresse à la description interne de la boîte noire. Si la description de la vue externe (boite noire) ne pose généralement pas de problème, la vue interne (l'architecture) peut quant à elle être réalisée selon plusieurs modèles de description.

Trois types de descriptions de fonctions sont identifiés et complémentaires : la description structurelle, la description comportementale, et la description fonctionnelle. Toutes ces descriptions sont mixtes, rendant complexes et inutile un classement strict et unique.

A. Description comportementale

Il s'agit d'une description indiquant le fonctionnement d'un système ou le comportement d'un modèle. Généralement réalisée sous la forme de processus, elle s'apparente au code procédural classique. Cette description ne repose pas sur une structure mais plus sur des algorithmes. Ce niveau de description peut être synthétisable pour être transformée automatiquement en structure matérielle.

B. Description structurelle (ou description matérielle)

Il s'agit d'une description schématique d'un système qui s'appuie sur des composants disponibles dans une bibliothèque et sur des signaux. Cette description est l'exacte représentation du schéma électrique du système. Elle définit la structure d'un modèle par un assemblage de composants, ou des boîtes noires. Ce type de description permet de créer la fonction voulue au niveau composant à l'aide de l'association d'éléments sous formes de portes logiques. C'est pourquoi la description structurelle est la seule à être implantable physiquement puisqu'elle fait référence au matériel de type porte logique élémentaire.

C. Description flot de données (ou description fonctionnelle)

Il s'agit d'une description indiquant comment un flot de données traverse un système. Le flot des sorties est exprimé en fonction du flot des entrées. Ce niveau de structure peut être synthétisable.

II.2.3 Langage de modélisation mixte multi-domaines

Un modèle mixte inclut des parties ayant un comportement dirigé par événements (parties logiques) et des parties ayant un comportement continu (parties analogiques) qui interagissent. La simulation mixte est, par exemple, une caractéristique incontournable de la conception des systèmes de puissance associant contrôle (algorithmique et électronique numérique), courants faibles (électronique) et courants forts (électronique de puissance et électrotechnique).

À l'aide de ces langages de modélisation de haut niveau, les différentes phases de conception peuvent être optimisées. Ces langages permettent de traiter indifféremment des modélisations logiques, analogiques ou mixtes au sein d'un même composant ou système. Par ailleurs, la philosophie de conception de ces langages et leurs jeux d'instructions en font des langages intrinsèquement multi-domaines qui gèrent les équations explicites, et, pour certains, implicites liées au fonctionnement d'un circuit. Un certain nombre de ces langages de modélisation ont été développés avec succès, comme : VHDL-AMS, Verilog-AMS, MAST, Modelica et la notation Bond Graph.

Le langage VHDL-AMS offre la possibilité de réaliser rigoureusement les interfaces logique ➔ analogique (L/A) et analogique ➔ logique (A/L) (Figure II.1) :

- L'interface (L/A) : est disponible par l'intermédiaire de l'instruction BREAK et des quantités implicites S'RAMP et S'SLEW.
- L'interface (A/L) : est disponible par l'intermédiaire du signal implicite *Q'ABOVE*.

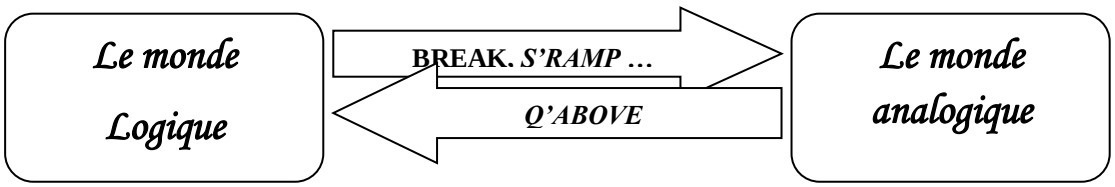

Figure II.1 : Interfaces L/A et A/L du langage VHDL-AMS.

II.2.3.1 VHDL-AMS

Afin de répondre aux différents besoins de l'électronique, la norme VHDL a dû évoluer. L'IEEE–DASC a créé la norme IEEE 1076.1-1999, plus connue sous la dénomination VHDL-AMS. Cette nouvelle norme est une extension de la norme IEEE 1076-1992. Elle a été réactualisée et complétée et définie actuellement par le standard : IEEE Std 1076.1 2007, qui a été publié le 15 novembre 2007 **[II.10]**.

La standardisation de VHDL-AMS est un facteur important pour la conception : la norme permet au concepteur d'accumuler ses expériences, d'utiliser l'expérience d'autres concepteurs et de profiter d'un nombre croissant d'outils qui supportent la norme. VHDL-AMS inclut toutes les propriétés du VHDL standard, avec en plus, la capacité de décrire les systèmes mixtes (analogiques et numériques) par le biais de modèles multi abstractions, multidisciplinaires, hiérarchiques à temps continu et à événements discrets à l'aide d'équations différentielles ordinaires. Des terminaux de connexion sont liés à des grandeurs physiques qui respectent implicitement les lois de Kirchhoff généralisées **[II.11]**.

VHDL-AMS permet de modéliser tout système dont le comportement peut être décrit par un ensemble d'équations différentielles algébriques et/ou ordinaires (ODE, ADE) qui ont le temps comme variable indépendante. Ces équations peuvent être écrites sous la forme : $F(x, dx/dt, \ldots, t)=0$, où x est le vecteur d'inconnues.

Le contributeur français incontournable autour de VHDL-AMS est «Yannick Hervé. Nous conseillons le lecteur par **[II.1]** pour une présentation exhaustive du langage.

Le schéma de la Figure II.2 montre les étapes de description de la méthodologie de conception dans l'environnement de modélisation VHDL-AMS, du cahier de charge vers les systèmes intégrés sur puce.

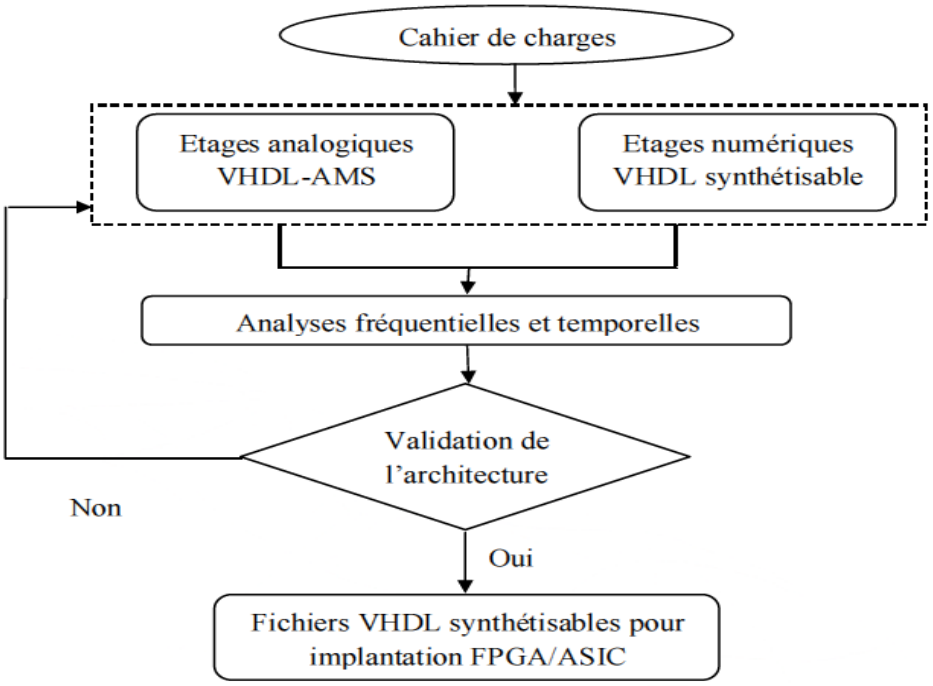

Figure II.2 : Description de la méthodologie de conception dans l'environnement VHDL-AMS.

II.2.3.2 Pourquoi le choix de VHDL-AMS ?

Les ressources de modélisation sont nombreuses dans une communauté large et active **[II.12]**. Les études qui ont été effectuées sur les caractéristiques principales pour différents langages de modélisation, nous ont permis d'apprécier que le langage VHDL-AMS est le plus adapté pour répondre à nos attentes et à nos besoins. Il est d'une très bonne lisibilité, il permet une haute modularité avec un typage fort (chaque objet doit être défini par un type) et supporte la généricité (le fait de pouvoir exprimer un modèle avec des paramètres qui ne seront connus qu'au moment de l'utilisation effective). Le standard VHDL-AMS permet de travailler en multi-équipes / multi-sites évitant l'obstacle à la communication entre domaines scientifiques, ainsi la norme autorise l'échange entre collaborateurs ou différents groupes d'ingénierie dans le cadre d'une bibliothèque IP (Intellectual-Property). Le fait que VHDL-AMS est un langage non propriétaire donne l'avantage d'un langage commun indépendant des fournisseurs et de la technologie **[II.13]**.

Le grand atout de ce langage est de permettre de modéliser et de simuler dans un environnement unique les différents éléments d'un système de puissance (simulation mixte multi-domaines), optimisant ainsi son étude et sa mise au point **[II.14]**. Cette souplesse d'emploi

permet aux concepteurs (électricien, mécanicien, …) de modéliser les parties d'un dispositif qui les concerne directement sans problèmes de dialogue avec les autres parties.

Un autre atout de VHDL-AMS est sa transparence : le concepteur possède la flexibilité pour modéliser ses propres systèmes comportementaux ou structurels et l'utilisateur possède la liberté de modifier les modèles pour les adapter à ses besoins. Ainsi, tous les modèles déjà créés sont archivables dans une bibliothèque (normalisée) pour être modifiés ou réutilisés plus tard.

VHDL-AMS permet d'associer des modèles de haut niveau et des modèles de bas niveau dans le même modèle global ayant une approche structurelle. L'association des méthodes de description permet de modéliser les systèmes mixtes **[II.15]**. La conception de systèmes selon les méthodes descendantes (top-down) et montantes (botom-up) peut être menée avantageusement avec VHDL-AMS en considérant sa capacité de modéliser à différente niveaux d'abstraction. Deux autres caractéristiques de VHDL-AMS sont le traitement des équations implicites sous forme d'équations différentielles non-linéaires, permettant de ne pas avoir à isoler les inconnues au moment de la formalisation du problème à résoudre en posant juste les contraintes, et à travers celles-ci, l'utilisation des lois de Kirchhoff généralisées (égalité des efforts, somme des flux égale à zéro), fondement des relations implicites entre les différents nœuds d'un système **[II.16]**.

Les modèles écrits en VHDL-AMS ne peuvent pas être exprimés avec des équations différentielles partielles *d./dx* avec *x* différent de t. Seules les dérivations temporelles sont acceptées par VHDL-AMS, ce qui rend délicat les modélisations géométriques ou à constantes reparties. Cette limitation a poussé un éditeur privé à développer une extension appelée VHDL-FD (FD pour Frequency Domain) au VHDL-AMS **[II.17]**.

II.3 Choix d'outils de simulation

Notre choix de langage s'est porté sur VHDL-AMS, en partie à cause de la diversité des simulateurs disponibles pour ce langage. Il existe plusieurs simulateurs pouvant manipuler des modèles écrits en VHDL-AMS, citons les simulateurs les plus importants: Smash de Dolphin-Integration, ADVance-MS et SystemVision de Mentor Graphics, SaberHDL de Synopsys, Simplorer de Ansys, Portunus de Adapted Solutions. Un simulateur VHDL-AMS doit disposer d'un compilateur permettant de traduire les modèles écrits en VHDL-AMS selon la norme IEEE-1076.1 en mode compréhensible par le simulateur. Par la suite, nous montrons les étapes du processus de simulation VHDL-AMS:

- **Description du système :** écriture du code du modèle,
- **Compilation :** analyses syntaxique et sémantique. Il permet la détection d'erreurs locales, qui ne concernent que l'unité compilée,
- **Stockage** dans la bibliothèque de travail,
- **Elaboration :** détection d'erreurs globales, qui concernent l'ensemble de l'unité de conception, développement de la hiérarchie, valeur des paramètres génériques, préparation des structure de données, préparation de structure de calcul prise en charge par le simulateur,
- **Simulation :** calcule le comportement du système après réglages des paramètres de contrôle,
- **Résultat :** tracé des chronogrammes.

II.3.1 Les simulateurs de première génération à code apparent

Ces outils comme hAMSter, ADVance MS, SMASH, …appartiennent pour la plupart à une génération de simulateurs dont le manque d'interface tend à devenir obsolète. En effet, ils ne s'inscrivent pas dans le cadre de l'accessibilité aux non spécialistes du langage. Il n'en reste pas moins que ce sont des outils de conception très puissants et intéressants pour les spécialistes à condition de pouvoir ensuite porter les modèles d'une plateforme de simulation à une autre. Dans cette catégorie de simulateurs le logiciel de hAMSter, qui n'est plus distribué aujourd'hui (racheté par Ansoft pour servir de base de développement de Simplorer), a quant à lui une très grande simplicité d'utilisation, mais une implémentation incomplète de la norme et un moteur de simulation moins rapide. Il est cependant gratuit, ce qui en fait un concurrent et une base très intéressante pour l'enseignement d'introduction. Le noyau de simulation de hAMSter a servi de base de développement pour celui du simulateur dernière génération d'Ansoft, Simplorer, propriété actuelle d'ANSYS.

II.3.2 SMASH 5

Pour simuler notre système, nous avons choisi le simulateur SMASH 5 développé par la société française Dolphin-Integration **[II.18]**. Au moment du choix il offrait le meilleur support de la norme VHDL-AMS et une polyvalence très importante quant aux différents langages qu'il supporte (SPICE, Verilog HDL, Verilog-A, C/SystemC, ABCD). Il fonctionne sur plates-formes UNIX et PC (Sun Solaris 7 - Linux RHL 7 to RHEL3 - Windows XP/vista). Durant la préparation de notre travail de doctorat nous avons utilisé plusieurs versions du simulateur SMASH 5 tel que : SMASH 5.12, 5.13, 5.15, 5.17, et dernièrement la version 5.19.

II.4 Principaux éléments du langage VHDL-AMS

II.4.1 Structure d'un modèle VHDL-AMS

Un modèle VHDL-AMS est constitué de deux parties principales : la spécification d'entité (mot clef : ENTITY) qui correspond à la vue externe du modèle et l'architecture de l'entité (mot clé : ARCHITECTURE) qui correspond à la vue interne du modèle. La Figure II.3 montre la structure d'un modèle VHDL-AMS **[II.1,II.10]**.

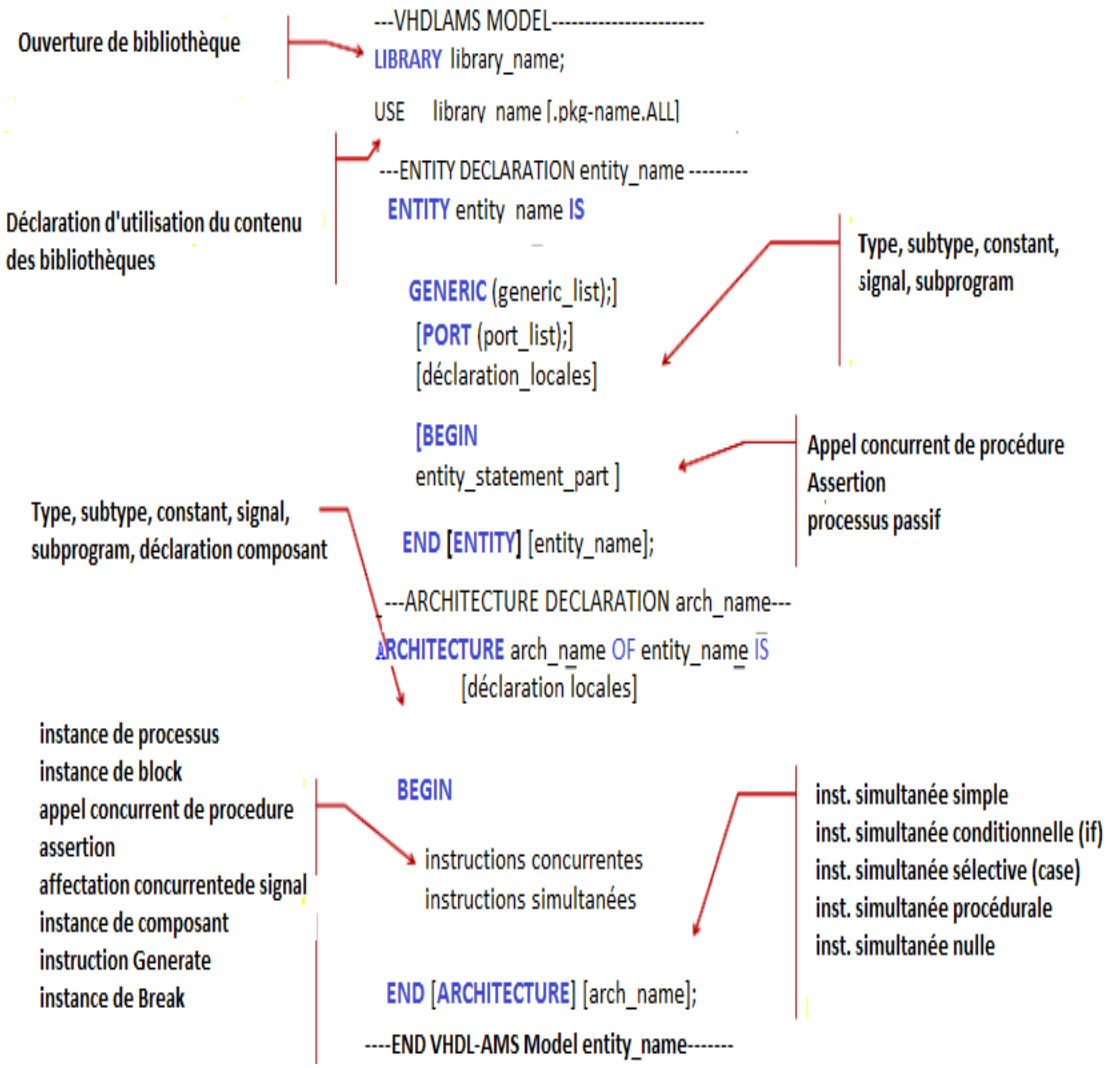

Figure II.3 : Structure d'un modèle VHDL-AMS.

II.4.1.1 L'entité

Nous pouvons comparer l'entité à une boîte noire où seuls les nœuds d'interconnexion sont visibles. Elle permet, après appel de bibliothèques utiles (mot clef : LIBRARY) et

spécification du contenu à exporter (mot clef : USE), de communiquer entre le monde extérieur et le modèle au moyen de deux objets : GENERIC et PORT.

- Les GENERICs sont des constantes (ou des variables statiques) à valeurs différées (les paramètres) qui peuvent être modifiés par la suite et servent à rendre le modèle plus général. Ces paramètres devront être fixés au moment de l'utilisation effective du modèle.
- Les PORTs sont les variables ou les nœuds dynamiques par lesquels l'entité pourra recevoir depuis et envoyer des informations vers son environnement (Figure II.4). 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. Les ports peuvent être de plusieurs classes :

  - Les ports de classe « signal » (mot clef : SIGNAL) définissent des canaux de communication directionnels (entrées « mode IN », sorties « mode OUT ») ou bidirectionnels « mode INOUT » modélisant des signaux logiques (à temps discret).
  - Les ports de classe « quantité » (mot clef : QUANTITY) définissent des points de connexions analogiques directionnels d'entrée « mode IN » ou de sorties « mode OUT » pour les informations analogiques orientées (temps continu, modélisation de type schéma-blocks).
  - Les ports de classe « terminal » (mot clef : TERMINAL) définissent des points de connexions analogiques pour lesquels les lois de conservation de l'énergie (lemmes de Kirchhoff pour les circuits électriques ou lemmes équivalents pour les systèmes non électriques) sont satisfaits (temps continu, modélisation de type conservative).

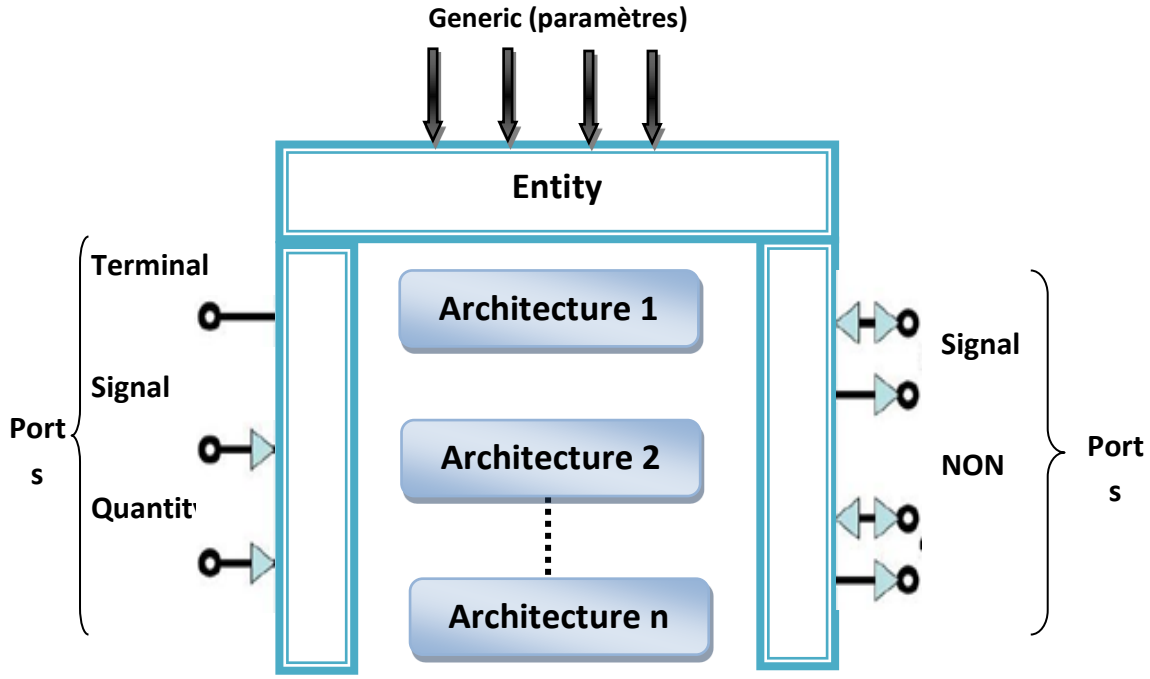

Figure II.4 : Vue général d'un modèle VHDL-AMS.

Une entité qui ne définit aucune déclaration est un banc de test, souvent appelée 'TestBench'. Le modèle interne ne pouvant être excité de l'extérieur doit être autonome pour son fonctionnement. La plupart des environnements de simulation ne peuvent prendre en charge que des modèles dont le point d'entrée est un 'TestBench'. La déclaration de l'entité d'un 'TestBench' est illustrée par le code suivant :

```
ENTITY TestBench IS
END ENTITY TestBench;
```

L'entité peut être écrite, compilée et mise à disposition avant de connaître l'architecture associée. Une fois l'analyse des besoins effectuée, on sait quels sont les ports d'entrée-sorties nécessaires. Cette souplesse peut servir de support à la Co-ingénierie, par exemple permettre le choix d'un boîtier pour le circuit alors que l'architecture n'est pas encore définie.

II.4.1.2 L'architecture

Une architecture définit le comportement et/ou la structure du système modélisé. Elle est reliée à une unique entité et hérite ainsi de toutes les déclarations faites à ce niveau (les constantes déclarées en GENERIC, les nœuds déclarés en PORT, …etc.).

L'architecture de l'entité permet, après une zone de déclaration locale, de définir le fonctionnement du modèle par l'intermédiaire d'instructions concurrentes permettant de manipuler l'information à temps discret, d'instructions simultanées mettant en jeu les valeurs à temps continu et des instanciations d'autres modèles, support de la description structurelle et la hiérarchie. L'intérêt de VHDL-AMS est que toutes ces instructions peuvent cohabiter, offrant ainsi un support à la multi-abstraction et à la conception mixte.

Pour une entité donnée, il peut y avoir plusieurs architectures qui permettent de lui associer différents types de description. Par contre pour une architecture donnée, il n'y a qu'une seule entité (Figure II.4). L'intérêt de disposer de plusieurs architectures pour le même modèle est extrêmement Important. Si une architecture comportementale a été validée, elle sert alors de référence (de cahier de charges) pour le reste de la conception. Quand le travail est plus avancé, on peut comparer alors, en Co-simulation l'adéquation des architectures.

II.4.1.3 Expressions de conditions initiales

Le mécanisme général d'initialisation des objets d'un modèle VHDL-AMS est que les objets prennent la valeur la plus à « gauche » de leur type ou la valeur qui est précisée dans leur déclaration. Une exception est faite pour les quantités qui sont initialisées à valeur initiale implicite « 0.0 », sauf en cas d'initialisation explicite. Ces initialisations sont valables à $T0^-$ (initialisation « informatique »)

La résolution numérique d'un ensemble d'équations différentielles est un exercice difficile qui peut ne pas converger vers une solution. On doit alors définir les conditions initiales des équations à T=0 pour que le simulateur puisse mener le calcul. L'initialisation effective est effectuée, en partant de l'initialisation « informatique », par un algorithme d'analyse permettant de calculer un point d'équilibre à T=0 (initialisation système ou analyse DC).

Comme on peut modifier les équations à utiliser en cours de simulation en gérant des discontinuités, la résolution n'est possible que si de nouvelles conditions initiales sont définies à chaque discontinuité. Donc, il faut pouvoir initialiser les quantités pour redémarrer un calcul jusqu'à la prochaine discontinuité. La norme définit de façon rigoureuse la gestion par l'utilisateur de cette contrainte très limitante dans d'autres langages. L'instruction « BREAK » permet de résoudre ce problème crucial d'initialisation, sa forme générale est la suivante :

```
BREAK [[FOR Q_name USE] Q_name => expression][ON S][WHEN condition];
```

L'initialisation au début du cycle de simulation pour le calcul du point de fonctionnement est effectuée par une instruction BREAK de forme simplifiée tel que :
Code de l'instruction BREAK (forme simplifiée) :

```
BREAK v => 0.0, s => 10.0;
```

Cette instruction système initialise (v) à 0.0 et (s) à 10.0 au temps = 0 seconde. Cette forme simplifiée du BREAK sans liste de sensibilité est activée une seule fois, au t = 0s, et suspendue jusqu'à la fin de la simulation. Les valeurs spécifiées remplacent l'initialisation « informatique » de façon prioritaire sur l'algorithme d'analyse DC.

L'instruction BREAK est utilisée aussi pour exprimer la discontinuité dans une simulation à temps continu et doit être utilisée pour synchroniser les noyaux de simulation à temps continu et à temps discret. La déclaration BREAK inclut une possibilité pour la spécification des nouvelles conditions initialisées pour les quantités spécifiques, qui seront appliquées juste au moment de la discontinuité.

II.4.2 Classes d'objets [II.10,II.15,II.19]

II.4.2.1 Terminaux et Natures

Le terminal est un objet VHDL-AMS utile pour la spécification de points de connexion pour lesquels les lois de conservation d'énergie sont satisfaites (lois de Kirchhoff pour l'électricité). Les terminaux peuvent être déclarés dans les interfaces des entités ou dans les zones de déclarations des architectures. Dans ce dernier cas, le terminal est local à l'architecture qui contient sa déclaration. Les terminaux ne définissent explicitement aucune valeur mais sont associés à des quantités pour former les DAE. Leur rôle donc est de faciliter la description des systèmes conservatifs. Les quantités associées sont appelées les quantités de branche (branch-quantity). Un terminal est associé à une « nature », qui représente un domaine d'énergie particulier (électrique, mécanique, thermique, …etc.), permettant de faire les vérifications de cohérence d'associations. Chaque domaine d'énergie est caractérisé par deux grandeurs liées à des effets physiques. Les grandeurs « ACROSS » représentent un effort et les grandeurs « THROUGH » représentent un flux. Ces deux classes de grandeurs permettent de définir la notion d'énergie et de puissance d'un système physique (Figure II.5).

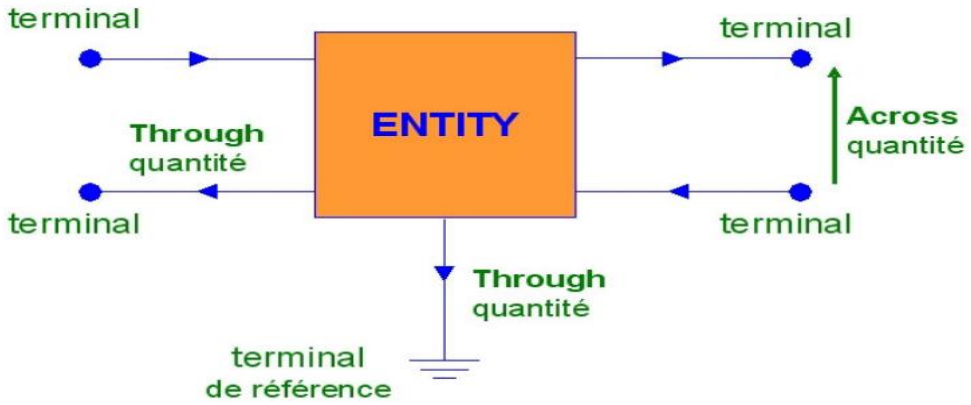

Figure II.5 : Les terminaux de l'entité.

Une nature est l'association de deux types réels et d'un identificateur, elle est associée aux mots clés effort "across" et flux "through", l'identificateur est associé au mot clé "reference" (tableau II.1).

Tableau II.1 : Flux et efforts relatifs à divers domaines physique.

Nature (domaine)	Effort (across)	Flux (through)
Electrique	Tension	Courant
Thermique	Température	Puissance (Heat_flow)
Mécanique de rotation	Angle de vélocité	Torsion
Mécanique de translation	Vélocité	Force
Magnétique	Force magnétique	Flux magnétique
Hydraulique	Pression	Flux (débit)

II.4.2.2 Quantités

Les quantités sont les objets qui transportent les signaux à temps continu. Une quantité peut être libre, de branche, de source ou implicite.

A. Une quantité libre (free_quantity)

Une quantité peut être déclarée dans un pot ou localement dans une architecture. Dans ce deuxième cas elle n'est disponible que localement. Elle peut être initialisée (sa valeur initiale par défaut est zéro), affectée et participe aux équations simultanées. La déclaration de quantité libre « Q » doit s'effectuer comme l'indique le code suivant :

```
QUANTITY Q : real := 2.0;   --Valeur initiale (à t0-) de Q est 2.0
```

B. Les quantités de branche (branch_quantity)

permettent d'associer des quantités à des terminaux (nœuds de connexion). Dans la déclaration d'une quantité de branche citée ci-dessous, (v) est la différence des efforts liés aux terminaux t_1 et t_2 qui est nommée quantité d'effort entre t_1 et t_2, utilise le mot clé ACROSS. Le point chaud de (v) est sur t_1. La quantité (i) est la quantité de flux qui s'écoule de t_1 vers t_2, utilise le mot clé THROUGH. Le type de (v) et (i) est défini par le type de la nature de terminaux t_1 et t_2. Si les terminaux t_1 et t_2 sont de nature electrical, alors (v) représente une tension et (i) un courant. Le code VHDL-AMS suivant représente la déclaration des quantités de branches.

```
QUANTITY v ACROSS i THROUGH t1 TO t2;
```

L'exemple suivant illustre la quantité de branche pour le domaine électrique (Figure II.6) :

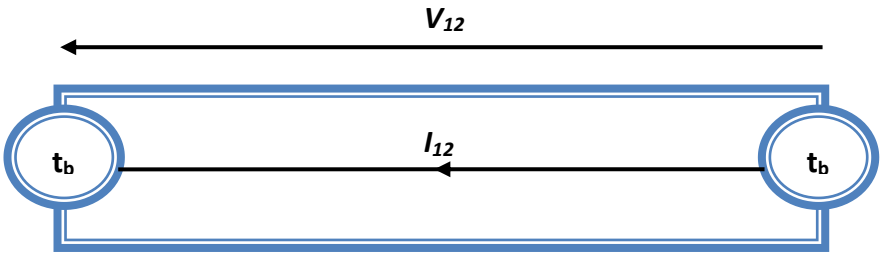

Figure II.6 : Illustration de la quantité de branche pour le domaine électrique

```
terminal tb1 , tb2 : electrical ;      --Définition des terminaux tb1 et tb2
quantity V12 accross                   -- quantité « entre » (Tension pour le domaine choisi)
I12 through tb1 to tb2 ;               --quantité « à travers » (Courant pour le domaine choisi)
```

C. Les quantités de sources (source_quantities)

représentent des sources utilisées pour les analyses spectrales et en bruit. L'analyse spectrale (analyse AC, ou petits signaux, dans les simulateurs de type Spice) se fait avec les sources spectrum après analyse du point de repos alors que l'analyse en bruit se fait avec les sources noise (en AC). Le code ci-dessous illustre la déclaration des quantités de sources.

```
QUANTITY v ACROSS i THROUGH p to m;
QUANTITY ac : real SPECTRUM 2.0, 0.0; --Magnitude ,Phase
QUANTITY nois_src : real NOISE 5.0 * amb_temp * K/R;
i == v/R + ac + nois_src ;
```

La dernière ligne du code précédent peut être remplacée par :

i = = v/R ; pour l'analyse TR (transitoire temporelle).

i = = ac ; pour l'analyse AC (harmonique).

i = = nois_src ; pour l'analyse noise.

D. Les quantités implicites (implicit_quantity)

Les quantités implicites sont des quantités qui n'ont pas besoin d'être déclarées, mais qui sont liées à d'autres quantités explicitement déclarées. La valeur initiale de quantité implicite est nulle, sauf en cas d'initialisation explicite.

Une première liste de quantités implicites est donnée ci-dessous. Soit Q une quantité explicitement déclarée :

- ***Q'dot*** représente la dérivée temporelle première de la quantité Q.
- ***Q'Integ*** représente l'intégrale de la quantité Q sur un intervalle de temps allant de zéro au temps courant.

La notation par attribut (***Q'dot***) est cumulative pour les dérivées des ordres supérieurs, par exemple: ***Q'dot'dot*** (resp. ***Q'dot'dot'dot***) représente la dérivée seconde (respectivement la dérivée troisième) de la quantité Q. Il en est de même pour les intégrales avec ***Q'Integ'Integ*** représente l'intégrale seconde de la quantité Q. Il existe d'autres quantités implicites (***'DELAYED, 'LTF, 'RAMP, 'SLEW,*** …etc.).

II.4.3 Sémantique de connexion

Les modèles compilés peuvent être instanciés dans des modèles de plus haut niveau. Ils sont alors interconnectés selon les besoins. Le comportement de ces associations sera déduit de la sémantique associée à ces connexions. Seules les quantités de mode OUT peuvent être branchées sur des quantités de mode IN. Dans ce cas, la valeur de la quantité sortante est recopiée sans modification sur la quantité entrante. En cas de connexion de terminaux, les équations de Kirchhoff généralisées sont implicites : les équations correspondantes sont ajoutées automatiquement au jeu d'équations constitutif du modèle. À chaque nœud (ou conjonction de terminaux) le simulateur s'arrange pour qu'en chaque point de simulation (ASP : les points temporels appelés points de solution analogique. il y ait une égalité des efforts et somme nulle des flux **[II.19]**.

II.4.4 Les instructions en VHDL-AMS [II.10,II.20]

II.4.4.1. Instructions séquentielles et concurrentes

La base d'un comportement dirigé par les événements est la notion de processus concurrents. Un processus (mot clef : PROCESS) en VHDL-AMS définit une portion de code dont les instructions sont exécutées en séquence dans l'ordre donné. Un processus (code ci-dessous) peut posséder sa propre zone de déclarations (subprogram, constant, attribute, … etc.) qui ne sont visibles qu'à l'intérieur du processus. Les instructions possibles dans un processus sont des instructions de contrôle (condition (IF), sélection (CASE), boucle (LOOP, WHILE, FOR)), d'affectation de variables et de signaux, de synchronisation (WAIT) et d'appel de procédure.

Code représente la déclaration d'instruction de processus :

```
[POSTPONED] PROCESS [( sensitivity_list )] [IS]
Declarations_locales
BEGIN
process_statement_part
END [POSTPONED] PROCESS;
```

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

II.4.4.2. Instructions simultanées

Les instructions simultanées (simultaneous-statement) constituent une classe particulière d'instructions permettant de décrire des équations différentielles algébriques linéaires ou non linéaires mettant en jeu des quantités, des constantes, des signaux, des valeurs littérales et des appels de fonction. Les instructions simultanées peuvent apparaître partout.

La forme la plus simple est l'instruction simultanée simple suivante :

```
expression1 == expression2 ;
```

Où (expression1) et (expression2) dénotent n'importe quelle expression VHDL-AMS légale dont l'évaluation produit une valeur réelle. Le code ci-dessous donne trois instructions simultanées simples équivalentes représentant l'équation constitutive de la résistance.

Les codes de l'instruction simultanée simple sont équivalents (en première approximation, car le mécanisme de TOLERANCE peut mener à des résultats de simulation de précisions différentes) :

```
v == R*i;          --v et i sont des quantités
i - v/R == 0.0;    --R est une constante
0.0 == v - R*i;
```

Plusieurs formes d'instructions simultanées sont disponibles en plus de la forme simple présentée ci-dessus. Il s'agit :

- Instruction simultanée conditionnelle, elle se construit à partir des mots clés « IF, USE »,
- Instruction simultanée sélective, elle se construit à partir des mots clés « CASE, USE »,
- Instruction simultanée procédurale « PROCEDURAL », permet de construire une équation simultanée en fonction des résultats d'un algorithme séquentiel.

II.4.5 Types d'analyses [II.15]

VHDL-AMS supporte différents types d'analyses, le simulateur utilisé doit fournir une indication explicite nommé DOMAIN. Les types possibles d'analyses sont les suivants :

II.4.5.1 Calcul du point de fonctionnement (DC)

Pour déterminer le point de fonctionnement, on utilise l'état « quiescent » (QUIESCENT_DOMAIN) illustré par le code ci-dessous. Ceci permet de contourner une limitation des simulateurs, qui ne prennent en compte la construction BREAK pour les conditions initiales.

Code de la déclaration de (QUIESCENT_DOMAIN) :

```
IF (domain = quiescent_domain) USE          --DC analysis
V == 0.0;
ELSE                                        --TR analysis
I == C * V'dot;
END USE;                                    --End Domain
```

En effet, il faut faire attention à la différence entre le mécanisme d'initialisation par l'instruction BREAK au sens informatique (à T0-) et au sens du système (à t=0, , ou l'utilisation du signal DOMAIN). Dans le premier cas, le concepteur donne des valeurs de départ pour commencer l'analyse DC préliminaire à l'analyse transitoire ou fréquentielle alors que dans le deuxième cas le concepteur impose la valeur de l'analyse DC. Notre travail de

modélisation des composants optoélectronique nous a permis de vérifier l'importance de l'initialisation des quantités libres au moment de la déclaration.

A. Analyse temporelle (TRANSITOIRE)

On utilise l'analyse temporelle (TIME_DOMAIN) pour déterminer le comportement transitoire temporel d'un système.

B. Analyse fréquentielle (AC, NOISE) :

L'analyse fréquentielle en petits-signaux (FREQUENCY_DOMAIN) , synonyme de linéaire, permet l'étude du régime permanent d'un circuit en ayant recours à des techniques de simulation simples et rapides. Le but est d'obtenir les caractéristiques en fréquence d'un circuit linéarisé autour du point de repos lorsqu'il est stimulé par des signaux sinusoïdaux de faibles amplitudes. L'analyse fréquentielle peut être l'analyse de bruit ou l'analyse harmonique.

II.4.6 Critère de solvabilité

La simulation analogique revient à résoudre un système d'équations à chaque pas de temps. Il faut s'assurer qu'à tout moment le modèle, au niveau de l'unité de conception, contient autant d'équations que d'inconnues. Le concepteur doit alors respecter obligatoirement le critère (nécessaire mais non suffisant) de solvabilité suivant :

NE = NQ

Où NE est le nombre d'équations simultanées, et NQ est le nombre d'inconnues constitué du nombre de quantités libres déclarées dans l'architecture + le nombre de quantité d'interface de mode OUT + le nombre de quantités de flux (THROUGH) - Nombre de quantités en mode OUT dans les interfaces des composants instanciés. L'intérêt de ce critère est qu'il peut être assuré localement, chaque unité de conception pourra être vérifiée par le concepteur et par le simulateur indépendamment du système dans lequel elle sera utilisée. Ce critère n'assure pas la CONVERGENCE qui dépend de valeurs dynamiques du système. Ce critère est obligatoirement nécessaire mais pas suffisant, un modèle qui n'assure pas ce critère ne passe pas la compilation et n'est donc pas simulé. Le code ci-dessous illustre un cas typique de non solvabilité, selon le critère ci-dessus. Pour l'architecture « non_correct » : on a NQ = 0 et NE = 1, par contre l'architecture « correct » assure la solvabilité : NQ = 1 et NE = 1.

Codes représente un modèle incorrect et un modèle correct d'une source de tension parfaite :

```
ENTITY vdc IS
```

```
    GENERIC (DC : real := 50.0);
     PORT (TERMINAL tp,tm : electrical);
END ENTITY vdc;
--Code ne sera pas compilé
ARCHITECTURE non_correct OF vdc IS
   QUANTITY v ACROSS tp TO tm;
   BEGIN
       V == DC;
END ARCHITECTURE non_correct;
--Code sera compilé et simulé
ARCHITECTURE correct OF vdc IS
QUANTITY v ACROSS i THROUGH tp TO tm;   --i ne sert qu'au critère
      BEGIN
          V == DC;
END ARCHITECTURE correct;
```

II.5 Le prototypage Virtuel Fonctionnel (PVF)

Différentes raisons peuvent entraîner de faire appel au prototypage virtuel et cela peut intervenir à différents niveaux d'abstraction du cycle en V. Les objectifs peuvent être l'étude d'une conception d'un produit ou de l'amélioration du dispositif que ce soit en terme de conception (réduire le cycle ou fiabiliser le circuit) ou en terme de technologie (amélioration des performances techniques du système). Les études commencent naturellement à partir des spécifications et d'un recensement des dispositifs existants. A partir des spécifications, le prototype virtuel peut demander plus ou moins de temps selon le niveau d'abstraction. Si par exemple, nous sommes à un niveau d'abstraction comportemental, la modélisation est rapide dans la mesure où les étages sont représentés par leurs fonction et non par leur description physique. Dans le cas de niveau d'abstraction physique, les temps de conception sont augmentés puisque le développement intègre la technologie. Dans tous les cas, les spécifications devront être établies en fonction du produit demandé et des dispositifs existants. En effet si la modélisation se fait sur la base de produits existants, les spécifications devront comprendre ses résultats réels. Par contre, les produits inexistants devront se baser sur les spécifications fonctionnelles.

L'implémentation d'un prototype virtuel **[II.21]** peut se faire par simulation ou Co-simulation (simulation avec plusieurs simulateurs). L'utilisateur peut dans ce type de prototypage faire cohabiter aussi bien des simulations de niveau RTL, que des simulations de niveau fonctionnel. L'avantage est sa flexibilité et son coût puisqu'il s'agit de simulation pouvant introduire des modèles de haut niveau. Le point faible de ce type de prototypage est qu'il est difficile de connecter un prototype virtuel avec l'environnement physique du système.

II.5.1 Avantage du processus PVF

Différents avantages techniques découlent naturellement de ce processus :

- Passé plus de temps en étude et innovation qu'en vérification ;
- Sortir des contraintes du développement incrémental ;
- Gérer les ruptures technologiques et vérifier simultanément le matériel et le logiciel ;
- Ecarter le prototypage matériel du chemin critique et réduire la durée et le coût des projets ;
- Augmenter la possibilité de réutilisation par la formalisation des savoirs et faciliter les échanges en interne ;
- Réduire les risques liés au développement du produit et réduire les coûts de prototypage ;
- Réduire les ressources en personnel et faciliter le travail avec les externes (partenaires, fournisseurs, sous-traitants) ;
- Augmenter la qualité et la fiabilité des produits et passer des connaissances formalisées de la sphère privée à la sphère entreprise ;

II.5.2 La description comportementale globale du système

Ce passage obligatoire de la conception d'un dispositif consiste à traduire les spécifications du système établies plus haut en un "cahier de charges simulable" qui reproduise le fonctionnement du système en établissant des relations entre les entrées et les sorties comme s'il s'agissait d'une "boîte noire".

Comme son nom l'indique, ce modèle ne fait que reproduire le comportement du système en utilisant les données disponibles le concernant. Dans le cas des spécifications fonctionnelles, la modélisation comportementale se situe au niveau de ce qui serait souhaité

par le groupe d'étude. Il se peut que devant des impératifs financiers ou technologiques, des solutions, donnant un résultat en désaccord avec les spécifications, s'imposent. Dans le cas où l'équipe de modélisation utilise des données contenues dans les feuilles de caractéristiques ou issues d'une série de mesures, la modélisation proposée peut refléter la réalité avec une précision extrême, mais sans aucun lien avec le fonctionnement interne du dispositif. Ce type de modèle descriptif, conçu pour simuler le fonctionnement du système dans une plage d'utilisation "normale", ne peut en aucun cas gérer les cas de figures pathologiques où le système est mis en déroute à cause de son fonctionnement interne ou d'une utilisation en dehors de son domaine de validité.

Les apports d'un modèle comportemental sont multiples. Il fournit tout d'abord une base à laquelle on pourra se référer pour valider toute simulation ultérieure des éléments de plus bas niveau constitutifs du système. Ensuite, un modèle comportemental est jusqu'à 1000 fois plus rapide à simuler **[II.22]** qu'un modèle bas niveau et rend bien compte de son fonctionnement standard. Les modèles comportementaux globaux sont donc des modèles descriptifs d'un système et constituent une étape indispensable qui servira de référence vers une modélisation de plus bas niveau.

II.5.3 L'étape structurelle

Il est rare que les objectifs d'un projet soient satisfaits par la simulation comportementale d'un dispositif. Il faut alors ôter le couvercle de la "boîte noire" du système pour en observer le fonctionnement et isoler les groupes physiques ou fonctionnels qui le constituent. Il ne s'agit pas d'un niveau de modélisation au sens de la simulation, mais plutôt au sens de l'expression du modèle. En effet, l'analyse structurelle établie les liens entre les blocs constituant le système, indépendamment de leur implémentation comportementale ou physiques. Sur le plan de la simulation, nous utilisons la représentation fonctionnelle qui substitue aux composants abstraits entre lesquels des liens ont été établis, dans le niveau structurel, leurs modèles comportementaux ou physiques, selon les besoins. Sur de gros systèmes, ou sur des systèmes multidisciplinaires, l'identification des éléments constitutifs d'un modèle structurel requiert l'intervention de spécialistes des domaines physiques ayant trait au dispositif. Il s'agit en effet, pour eux, de choisir les sous-ensembles à isoler, en fonction des interactions possibles entre eux et des niveaux de modélisation que l'on souhaite associer à chacun selon les objectifs recherchés. Effectivement, chaque élément du modèle

fonctionnel peut à son tour faire l'objet d'une analyse structurelle, et ainsi de suite, jusqu'à descendre au niveau de modélisation physique des différents composants.

II.5.4 La description physique des composants

Lorsque l'on souhaite avoir une modélisation prédictive très fine d'un composant, il est nécessaire de descendre très profondément dans ses mécanismes internes pour faire ressortir les équations qui gouvernent son comportement et permettent d'appréhender les effets parasites les plus fins. Ces modèles, qui sont en général beaucoup plus lents que leurs homologues comportementaux, du fait du nombre constitutif d'équations complexes, ont pour eux la "productivité" et la précision de leurs résultats. De tels modèles sont en effet capables de traduire les comportements atypiques ou émergents qui échapperaient totalement à une modélisation de plus haut niveau. La mise au point de ces modèles nécessite l'intervention de tous les domaines de compétences.

Les spécialistes de la physique doivent développer les modèles mathématiques rendant compte au mieux la subtilité du fonctionnement des composants, alors que les spécialistes des langages tenteront de trouver la solution informatique la mieux adaptée au traitement du modèle physique. Une fois ces étapes explorées, nous disposons d'une bibliothèque de modèles VHDL-AMS qui permette de décrire tout ou partie d'un système. Il reste ensuite, éventuellement, à assembler ces différentes briques pour tester et valider notre modélisation.

II.6 La modélisation comportementale

Le niveau de modélisation comportementale représente la première description différenciée (en parties numérique, analogique ou autre) du système global. Il a une position centrale dans le flot de conception des systèmes mixtes, indispensable à la phase de conception descendante comme à la phase de validation ascendante.

D'après **[II.23]**, un modèle comportemental peut être défini comme étant une représentation abstraite ou une description d'un système physique dont on ne conserve que les aspects essentiels à une certaine utilisation. Ces aspects ou comportements expriment des relations de cause à effet, des algorithmes, des processus ou des équations définissant des relations entre variables qui représentent des grandeurs physiques d'entrées/sorties. Ainsi, un modèle comportemental doit répondre aux exigences suivantes **[II.1,II.24]** :

- Une description fidèle des comportements choisis pour la représentation du système lorsque celui-ci est soumis aux stimuli relatifs à son utilisation.

- Une simulation rapide et fiable pour les différentes conditions d'utilisation et les différents modes de simulation.
- Une compatibilité des nœuds d'entrées/sorties avec les autres composantes du système global.
- Une transportabilité du modèle pour permettre sa réutilisation en tant qu'IP. Cela implique l'utilisation d'un langage de description standard et la possibilité d'adapter le modèle à d'autres utilisations similaires, par l'intermédiaire de paramètres génériques.

En résumé, l'obtention d'un modèle comportemental doit réaliser un compromis entre rapidité de simulation et précision de la description. Ce compromis dépend des conditions d'utilisation du modèle telles qu'elles sont définies par les plans de modélisation et de validation élaborés par le concepteur. Ainsi, si le modèle est destiné à être utilisé lors de la phase de conception descendante (Top-Down), la rapidité de la simulation prime généralement sur la précision de la description. Pour la phase de validation ascendante, c'est un modèle extrait qui est utilisé **[II.25-II.26].**

Ces deux types d'utilisation impliquent deux méthodes différentes pour l'obtention des modèles comportementaux **[II.27]** :

- Une approche ascendante où les modèles sont extraits à partir d'une description hiérarchiquement inférieure.
- Une approche descendante où les modèles sont génériques et principalement destinés aux premières phases du flot de conception.

II.6.1 Obtention d'un modèle comportemental par la méthode ascendante

La méthode ascendante consiste en l'élaboration d'un modèle comportemental en partant d'un système existant ou d'une modélisation de niveau inférieur. Les modèles ainsi obtenus sont surtout destinés à être utilisés lors de la phase de validation d'un système complexe. En effet, ils doivent permettre une simulation bien plus rapide qu'une description niveau transistor par exemple, tout en gardant les caractéristiques essentielles du circuit modélisé. Pour les systèmes ou circuits analogiques, deux types d'extraction sont classiquement envisagés **[II.28]**:

- L'exploitation d'une description au niveau diode laser. Elle consiste en la réduction du système d'équations qui caractérise le circuit. Cette simplification doit permettre une simulation plus rapide du système, tout en gardant ses caractéristiques de transfert (entre les variables d'entrées et de sorties) à un niveau de précision acceptable **[II.27,II.29]**.

- L'exploitation des résultats de simulation du modèle au niveau diode laser. Dans ce cas, la modélisation consiste en la recherche de fonctions mathématiques approchées, permettant de décrire les variables de sorties**[II.28]**.

Pour les composants de la ligne de transmission choisie, l'obtention des modèles comportementaux provient de trois sources principales:

- L'exploitation des résultats de simulations optoélectronique .
- L'exploitation de mesures réalisées sur des composants optoélectronique existantes.
- L'extraction d'équations simplifiées à partir de schémas électriques équivalents aux composants optoélectronique à modéliser.

Notons que les modèles de composants optoélectronique prennent généralement en compte des comportements hautement non-linéaires ce qui nécessite des techniques d'extraction complexes.

II.6.2 Obtention d'un modèle comportemental par la méthode descendante

La deuxième famille de modèles comportementaux est composée de modèles dits génériques, car généralement décrits indépendamment d'un composant ou circuit de référence. L'approche qui conduit à l'obtention de ce type de modèle a pour base la fonction première de l'objet à modéliser. Cette démarche présente l'avantage de produire des modèles plus rapides à simuler, plus paramétrables et donc plus transportables que ceux obtenus par la méthode ascendante présentée ci-dessus **[II.30]**. Néanmoins, la précision des résultats de simulation est amoindrie.

L'obtention de ce type de modèle est assez intuitive. Une méthodologie systématique a été proposée par **[II.27]**. Elle propose de décomposer le modèle en trois parties comme indiqué sur la Figure II.7.

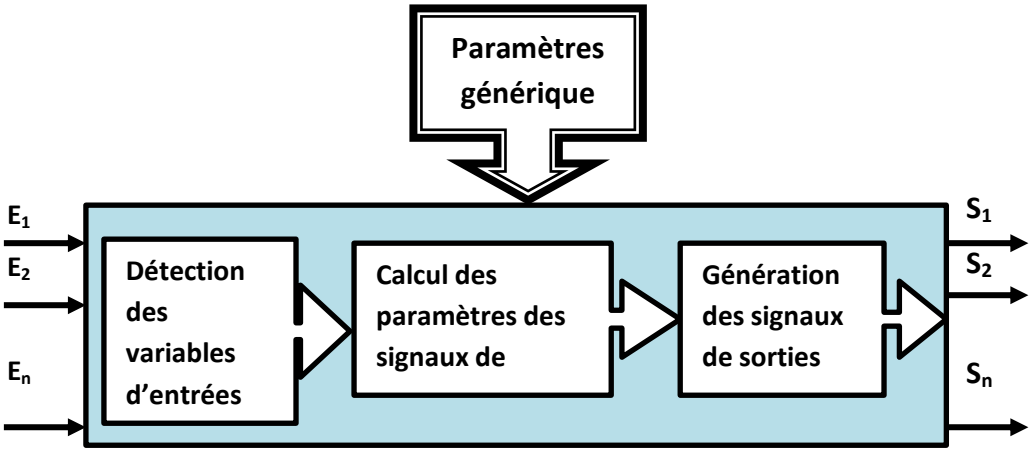

Figure II.7 : Structure fondamentale d'un modèle obtenu grâce à l'approche systématique.

L'objectif est de mettre en relation les entrées/sorties en fonctions des paramètres génériques et de la caractéristique de transfert. Pour cela, le modèle doit détecter les variables d'entrées, appliquer la fonction de transfert qui permet de calculer les paramètres de sorties et enfin, générer les signaux de sorties. L'étape de détection des variables d'entrées permet d'isoler la ou les composantes du signal d'entrée sur lesquelles s'applique la fonction de transfert. A titre d'exemple on cite la tension ou le courant en entrée, l'amplitude d'un signal, sa fréquence ou sa phase. La deuxième étape consiste en la modélisation de la relation entre les paramètres d'entrées détectés et les paramètres de sorties. Cette relation peut être mathématique, algorithmique ou composée de tout autre type d'instructions que permet le langage de modélisation. Elle doit évidemment prendre en compte les paramètres génériques du modèle. Enfin, la dernière étape doit assurer la génération des signaux de sorties en fonction des modifications survenues lors de l'étape précédente.

Cette famille de modèles comportementaux est destinée à être utilisée lors des premières phases du flot de conception Top-Down. Le caractère générique de ces modèles leur permet de constituer des bibliothèques d'IP réutilisables pour différents projets de conception de systèmes mixtes. Il existe d'ailleurs plusieurs organismes qui partagent leurs modèles comportementaux comme l'association BEAMS **[II.31]**, les bibliothèques Commlib de Mentor Graphics, la bibliothèque fournie par Cadence ou celle de l'institut Fraunhofer de Dresde **[II.32-II.33]**.

II.7 Conclusion

Chaque langage permet de modéliser le comportement d'un composant avant que celui-ci ne soit construit réellement. Comme énoncé précédemment, notre choix s'est porté sur VHDL-AMS en raison de la diversité des simulateurs disponibles pour ce langage. Dans ce chapitre, nous avons reporté en détail les résultats de recherches sur VHDL-AMS pour connaitre comment utilisé ses constructions et ses instructions fondamentales. Le langage de description du matériel normalisé VHDL-AMS, nous a permis de modéliser un système de transmission optique multi-domaines avec différents niveaux d'abstraction en utilisant le simulateur SMASH. Pour écrire les codes des différents composants, il faut choisir la méthode qui sera utilisé dans le chapitre suivant pour que les résultats soient justes, dans notre travail on a choisi la méthode « Top Down ».

II.8 Références bibliographiques

[II.1] Y. HERVE, « VHDL-AMS, Applications et enjeux industriels », Dunod, Paris, 2002.

[II.2] B. Delinchant, L. Gerbaud, « Capitalisation et Réutilisation de modèles », G2ELab, équipe MAGE, ALSTOM Tarbes, Février 2008.

[II.3] SEEDS, LIG(SIGMA), G-SCOP, « Première maquette DIMOCODE », Présentation DIMOCODE, YACS, Janvier 2008.

[II.4] T. Riesgo, Y. Torrojaand, E. Torre, « Design Methodologies Based on Hardware Description Languages », IEEE Transaction on Power Electronic, pp: 3-11, Vol. 46, No. 1, February 1999.

[II.5] IEEE Computer Society, "IEEE Standard for Verilog Hardware Description Language", IEEE Std 1364-2005, pp. 590, New York- USA, 2006.

[II.6] Homepage of the Verilog modeling language. http://www.verilog.com/

[II.7] EEEComputer Society, "IEEE Standard VHDL Language Reference Manual", Amendment 1: Procedural Language Application Interface, IEEE Std 1076c-2007, pp. 226, New York-USA, 2007.

[II.8] U. Heinkel, M. Padeffke, W. Haas, « The VHDL Reference: a practical guide to computer-aided Integrated circuit design including vhdl-ams », Wiley, pp. 420, Chichester-New York, 2000.

[II.9] L. Hoang , "Introduction à MATLAB et Simulink ", Département de génie électrique et de génie informatique Université Laval Québec, CANADA Septembre 1998.

[II.10] IEEE Computer Society, "IEEE Standard VHDL Analog and Mixed-signal Extensions", IEEE Std 1076.1-2007, pp. 342, New York-USA, 2007.

[II.11] Homepage of the IEEE standard VHDL-AMS, Http://www.eda.org/vhdl-ams/

[II.12] Homepage of the Systems'Vip, « Systems'Virtual Prototyping Company», http://www.systemsvip.com

[II.13] Y. HERVE, « La dynamique VHDL-AMS, Enjeux et difficultés de la transformation de modèles », Séminaire LAAS-TOOLSYS, Février 2006.

[II.14] P-R. Wilson, J-N. Ross, A-D. Brown and A. Rushton, « multiple domain behavioral modeling using VHDL-AMS », Proc. IEEE ISCAS, pp: 644-647, 2004.

[II.15] R. Frevert, J. Haase, R. Jancke, U. Knöchel, P. Schwarz, R. Kakerow, M.Darianian, « Modeling and Simulation for RF System Design », Chapter 6: Introduction to VHDL-AMS, pp: 51-125, Springer US, 2005.

[II.16] Y. HERVE, " Prototypage Virtuel Fonctionnel et VHDL-AMS", Congrès CNFM'06, Systems'Vip, Novembre 2006.

[II.17] Logiciel Rincon de Ridgetop EDA Software.www.ridgetop-group.com/docs/rincon_datasheet.pdf.

[II.18] Dolphin Integration, «SMASH», http://www.dolphin.fr/medal/smash/smash_overview.php.

[II.19] Y. HERVE, « Extension AMS du langage VHDL pour l'électronique de puissance », Techniques de l'Ingénieur, Ref' D3067, pp. 17, Paris, 2005.

[II.20] A. Vachoux, « Modélisation de Systèmes Analogiques et Mixtes, Introduction à VHDL-AMS », Laboratoire de Systèmes Microélectroniques STI-LSM.

[II.21] C-A. VALDERRAMA « Prototype virtuel pour la génération des architectures mixtes logicielles/matérielles » Thèse de doctorat de l'institut national polytechnique de grenoble, Octobre 1998.

[II.22] B. MIRAMOND, « Exploration architecturale», http:// www.lami.univ-evry.fr/~miramond/CV/CVlg/node20.html.

[II.23] A. VACHOUX , "Modélisation de Systèmes Analogiques et Mixtes -Introduction à VHDLAMS", notes de cours 2003 EPFL. http://lsmwww.epfl.ch/design_languages/Model_Sys_Mix/Documents/modelmix03.pdf.

[II.24] R. Frevert, J. Haase, R. Jancke, U. Knöchel, P. Schwarz, R. Kakerow and M. Darianian, "Modeling and Simulation for RF System Design", Springer, First edition, 2005.

[II.25] J. Hartung, H.J. Wassener, G. Tränkle, M. Schröter, "Hot topic session: RF Design Technolohy for Highly Integrated Communication Systems", IEEE, 2003.

[II.26] K-S. Kundert, "Introduction to RF simulation and its application", IEEE Journal of Solid-State Circuits(http://www.kenkundert.com/pubs.html), Vol. 34, No. 9, Septembre 1999.

[II.27] A. Fakhfakh, "Contribution à la modélisation comportementale des circuits radio-fréquence", thèse de l'université de Bordeaux I, Janvier 2002.

[II.28] J.R. Phillips, "Projection-based approaches for model reduction of weakly nonlinear time-varying systems", IEEE Transactions onComputer Aided Design of Integrated Circuits and Systems, Vol. 22,No. 2, Février 2003.

[II.29] G. Casinovi, A. Sangiovanni-Vincentelli, "A Macromodeling Algorithm for Analog Circuits", IEEE Transactions on Computer Aided Design, Vol. 10, No. 2, 1991.

[II.30] G.R. Boyle, B.M. Cohn, D.O. Pederson, J.E. Solomon, "Macromodeling of integrated circuit operational amplifiers", IEEE Journal of Solid-State Circuits, SC 9, 1974.

[II.31] N. Milet-Lewis, G. Monnerie, A. FakhFakh, D. Geoffroy, Y. Hervé, H. Lévi, J.J Charlot, " A VHDL-AMS library of RF blocks models", BMAS 2001, Santa Rosa, Octobre 2001.

[II.32] Association BEAMS (Behavioural Modelling of Analogue and Mixed Systems). https://www.listes.u-bordeaux1.fr/wws/info/beams_members.

[II.33] R. KHOURI, « Modélisation comportementale en VHDL-AMS du lien RF pour la simulation et l'optimisation des systèmes RFID UHF et micro-ondes », Thèse de Doctorat, L'INP Grenoble ; le 28 Mai 2007.