

I.1 Introduction :

Le modèle d'un système est une représentation de son comportement à l'aide de laquelle le simulateur comprend et procède à des calculs. 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. De plus, ce comportement doit être compréhensible par le simulateur [3].

En plus du simulateur de circuits SPICE, il est apparu sur le marché international un langage de description matérielle : le VHDL-AMS qui répond à des besoins analogiques, numériques et mixtes.

Dans ce chapitre, nous allons commencer par présenter une initiation détaillée au langage VHDL-AMS. Une méthode de modélisation par le biais de ce langage est présentée

I.2 Simulation des circuits électroniques analogiques :

Simuler un circuit à l'aide d'un programme de CAO revient donc à remplacer son étude sur maquette par une étude sur ordinateur. Ceci implique que l'on puisse schématiser le système ou ses éléments par des modèles les plus proches possibles de la réalité. Il faut ensuite formuler les équations du système, puis les résoudre par des moyens analytiques ou numériques.

I.2.1 Formulation des équations pour simuler un circuit :

Les variables électriques inconnues dans un réseau sont les tensions des nœuds, les courants de branches, et les énergies stockées par les éléments réactifs. Les équations résolues par le programme d'analyse sont formulées à partir de la description réseau. En fait, toute méthode de mise en équation revient à déterminer le système algèbre-différentiel des variables électriques, déduit des lois élémentaires d'Ohm et de Kirchhoff.

I.2.2 Résolution des équations d'un circuit :

Les méthodes de résolution des équations d'un circuit analogique sont subdivisées en trois niveaux :

- 1- Analyse d'un circuit linéaire en régime statique.
- 2- Analyse d'un circuit non-linéaire en linéarisant le circuit et en utilisant l'analyse du circuit linéaire en régime statique.
- 3- Analyse d'un circuit en régime dynamique en convertissant le circuit considéré en un circuit indépendant du temps.

Quelle que soit la méthode utilisée pour mettre en équations le réseau étudié, l'analyse en continu et en transitoire conduit à un système d'équations différentielles non linéaires. La simulation analogique est beaucoup plus complexe que la simulation logique et requiert ainsi plus de ressources (temps de calcul, mémoire). De plus, elle implique la résolution d'équations différentielles et algébriques linéaires et non linéaires. Les solutions sont des tensions entre les nœuds du circuit et les courants dans les branches du circuit.

Trois types d'analyse peuvent être considérés :

a) L'analyse temporelle (transient analysis) :

Cette analyse calcule les réponses temporelles du circuit (tensions et courants en fonction du temps) relativement à un ensemble de stimuli (excitations).

b) L'analyse DC (direct current) :

Elle calcule l'état du circuit pour un ensemble de stimuli (excitations) fixés après un temps infiniment long (steady state). L'analyse DC est utile pour calculer le point de repos, ou la polarisation du circuit, la fonction de transfert, la résistance d'entrée et de sortie du circuit, les sensibilités de variables de sortie en fonction des paramètres du circuit... [2]

c) L'analyse AC (alternative current) :

Calcule les réponses fréquentielles du circuit en régime petits signaux. Un signal alternatif est alors appliqué, autour du point de repos du circuit. L'analyse AC est utile pour calculer les fonctions de transfert (par exemple le gain en tension du circuit) en fonction de la fréquence et des conditions de polarisation du circuit. Elle est aussi utile pour analyser l'influence du bruit ainsi que la détermination des caractéristiques de distorsion du circuit [2].

I.2.3 Convergence et stabilité :

Les problèmes de convergence peuvent être inhérents au circuit étudié dans le cas d'un circuit ayant plusieurs états d'équilibre (ex. bascule) ou n'ayant pas d'état d'équilibre (ex. oscillateur) ou bien inhérents à l'algorithme utilisé. Des problèmes de débordement peuvent également apparaître dans le cas des caractéristiques non linéaires de forte croissance (ex. exponentielle). La stabilité d'un algorithme d'intégration (ex : analyse en transitoire) est liée à la valeur du pas d'intégration. Une bonne stabilité et une faible erreur locale sont en général obtenues grâce à un pas suffisamment petit. Cependant, pour que le calcul se déroule rapidement, il faut utiliser un pas relativement grand. Afin d'obtenir le meilleur compromis précision/rapidité de calcul, le programme doit, à chaque étape, ajuster le pas à une valeur optimale. Dans ce but, il a été

nécessaire de développer des algorithmes d'intégration de stabilité maximale et a pas dit auto-adaptatif.

I.3 Modélisation et simulation SPICE :

SPICE (Simulation Program with Integrated Circuit Emphasis), est un simulateur des circuits électriques pour des analyses en continu **DC** non-linéaires, temporelles non-linéaires **TRAN**, et fréquentielles **AC** linéaires et paramétriques.

Pour l'analyse **DC**, **SPICE** détermine le point de fonctionnement en mettant les capacités en circuit ouvert et les inductances en court-circuit. **SPICE** procède par itérations numériques pour résoudre les équations non-linéaires.

Dans l'analyse temporelle, les conditions initiales sont déterminées automatiquement par l'analyse **DC**, **SPICE** fait le calcul pour chaque nœud du circuit en fonction du temps. C'est une analyse de grands signaux où il n'y a pas de restriction sur l'amplitude du signal d'entrée. En ce qui concerne l'analyse fréquentielle **AC**, **SPICE** fait le calcul des valeurs complexes des tensions de chaque nœud du circuit considéré [4].

I.3.1 Limitations et contraintes :

Le simulateur **SPICE** est considéré en réalité comme un standard pour l'analyse des circuits. Très vite il est devenu l'outil le plus efficace d'aide à la conception. Pourtant, il comporte des limitations dans certains domaines :

a) Modélisation mixte :

Le simulateur **SPICE** est à temps continu, donc le modèle conçu doit être à temps continu. **SPICE** ne peut pas supporter les représentations discrètes, et en conséquence, il n'est pas adapté pour la modélisation mixte, (temps continu et discret à la fois) sauf au moyen d'une macro-modélisation lourde [2].

b) 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 (structurel) 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 composants internes.

c) Transmission de données :

SPICE ne supporte que des systèmes conservatifs comme par exemple les circuits électriques qui obéissent aux lois de Kirchhoff (loi des nœuds et loi des mailles). En ce qui concerne les flots de données (non conservatifs), une des façons de les représenter utilise le temps discret, et cette représentation n'est pas supportée par SPICE [2].

d) 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 élaborés sous SPICE sont en général assez complexes et l'utilisateur ne peut alors pas contrôler les équations primitives considérées [5].

I.4 Langage de description matériel HDL (Hardware Description Language) :

Contrairement à un langage informatique, le langage HDL ne vise pas forcément une exécution. Il sert à décrire du matériel, comme son nom l'indique, avec pour objectifs la spécification, la modélisation, la simulation, la documentation, la synthèse logique, la preuve formelle ou l'extraction (LVS : "lay-out versus schématique"),...

Dans un langage informatique, l'information est transportée par des variables d'usage dynamique alors que dans un HDL, l'on définit des signaux qui servent de façon statique à modéliser des équipotentiels. L'information associée à ces signaux peut être datée et c'est l'histoire de celle-ci qui permet de construire les chronogrammes. Les variables, avec un langage informatique, sont traitées par des procédures (sous-programmes ou fonctions) qui sont appelés, exécutés et oubliés. La mémoire nécessaire (avec les variables) est allouée dynamiquement. Dans un HDL, l'information est transformée par des composants qui sont utilisés de façon statique et reliée par des signaux. Ils permettent de modéliser des composants matériels et n'ont aucune raison de disparaître. Dans le modèle HDL le traitement du temps fait partie intégrante de la sémantique des modèles [6].

I.4.1 Le langage VHDL (Very high scale integrated circuit Hardware Description Language):

Le langage VHDL est un standard IEEE (IEEE 1076-1993) pour la modélisation, la simulation et la synthèse des systèmes matériels logiques (HDL - Hardware Description Language). Il est aujourd'hui très largement utilisé et est supporté par tous les environnements d'aide à la conception de circuits et de systèmes électroniques (EDA - Electronique Design

Automation) [7]. Le VHDL est un puissant langage de description des circuits d'électronique numérique. Avec le VHDL, il est possible de simuler et de synthétiser des circuits numériques pour différentes technologies. Toutes entités déjà créées sont archivables dans une librairie pour être modifiées ou réutilisées 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 [8].

I.4.2 Le langage VHDL-AMS (VHDL –Analog and Mixed Signal):

Le langage VHDL-AMS est également un standard IEEE (IEEE 1076.1-1999). Il a été développé comme une extension du langage VHDL pour permettre la modélisation et la simulation des circuits et des systèmes analogiques et mixtes logiques-analogiques [7].

Le VHDL-AMS est un langage de description hiérarchique qui permet de simuler des systèmes continus et mixtes (c'est un standard dont l'apparition a rendu possible le couplage du niveau HDL (multi abstraction) entre les domaines analogiques et les domaines numériques en terme de modélisation). Le langage VHDL-AMS permet la description et la modélisation des systèmes conservatifs continus et mixtes (continus/discontinus) aussi bien que non-conservatifs continus et mixtes (continus/discontinus). Les algorithmes de simulation ne font pas partie du langage. Il permet la modélisation sur trois niveaux : comportemental, fonctionnel et physique. Cette modélisation peut être appliquée dans différents domaines électriques et non électriques (multi-technologie : thermique, mécanique, hydraulique,..... etc.). Les modèles écrits en VHDL-AMS autorisent tous les types d'analyse : DC, transitoire, petits signaux,.....etc. Les circuits analogiques modélisés sont décrits par des systèmes d'équations ordinaires différentielles algébriques (par rapport au temps): EDA (Equations ordinaires Différentielles Algébriques). Il supporte les transformations de Laplace et en Z. Le VHDL-AMS sert à décrire des systèmes mixtes continus/discrets. Le caractère continu de ces systèmes est décrit par des EDA_S où le temps est considéré variable indépendante. La norme VHDL-AMS ne décrit pas d'algorithme de résolution des EDA_S , mais fournit une notation pour ces EDA_S , il suffit que VHDL-AMS décrive le type de système d'EDA décrit par le modèle. Le VHDL-AMS ne fait que caractériser les résultats que doit obtenir l'algorithme de la solution appelé solveur analogique.

Dans le domaine numérique, les inconnues des EDA_S signaux et variables, obtiennent leurs valeurs par affectation séquentielle. Mais dans le domaine analogique les inconnues des EDA_S sont des fonctions analytiques du temps, c.-à-d. individuellement continues avec un nombre fini de discontinuités. Le solveur trouve des solutions pour toutes les inconnues en fonction du temps en convertissant d'abord, pour des valeurs précises du temps, la partie

différentielle des EDA_5 en équations aux différences en utilisant des méthodes adéquates, puis, en résolvant simultanément les équations aux différences [4].

Remarque : Synchronisation des noyaux :

Pour la simulation mixte, il faut que les noyaux numérique et analogique puissent se synchroniser. Dans la plupart des applications, chaque noyau gère sa propre horloge mais doit conserver des liens avec l'autre pour assurer la cohérence de l'évaluation. L'action du noyau numérique sur le noyau analogique, correspond à un événement sur un signal. Elle doit pouvoir provoquer une évaluation analogique en utilisant l'instruction `BREAK ON (s)`. L'action du noyau analogique sur le noyau numérique est réalisée en fabriquant un événement à partir d'une quantité avec l'instruction `Q'ABOVE (ref)`.

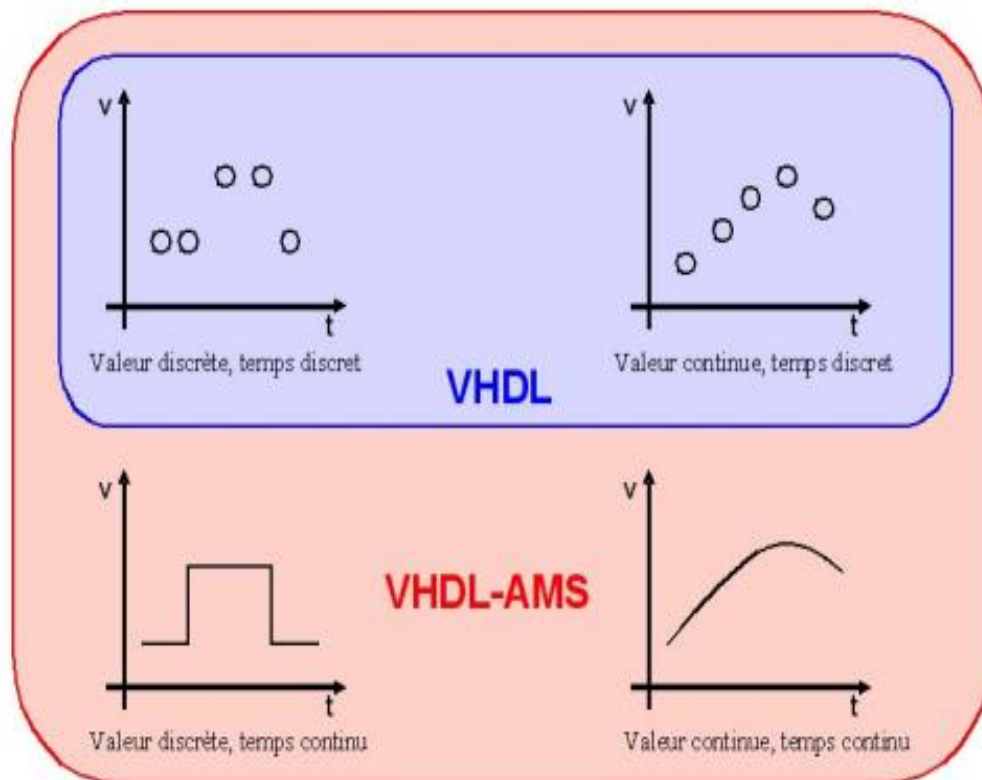


Figure I.1 : Couverture de VHDL-AMS

I.4.3 Avantage de VHDL-AMS :

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

de nombreuses heures de travail supplémentaires. VHDL-AMS est quant à lui un produit non propriétaire et normalisé par l'IEEE qui tend à être reconnu par le plus grand nombre. L'utilisation généralisée de ce langage facilite la communication entre les différents domaines scientifiques grâce à son approche multi domaines native qui permet aussi bien à un électronicien qu'à un mécanicien ou même un chimiste de modéliser la partie d'un dispositif qui le concerne directement sans problèmes de dialogue avec les autres parties. La grande force de ce langage est de permettre la simulation mixte en autorisant aussi bien les modélisations à temps continu (analogiques) qu'à événements discrets (logiques) ou mélangeant les deux. A cette flexibilité d'emploi s'ajoute la possibilité pour les concepteurs d'aborder leurs modèles à différents niveaux d'abstraction. En effet, VHDL-AMS propose des mécanismes permettant de gérer aussi bien les abstractions comportementales (c'est la fonction réalisée par le système qui est modélisée et non sa physique), que les abstractions structurelles (le système est divisé en sous-ensembles qui peuvent eux-mêmes être modélisés au moyen de différentes abstractions, ...) ou bien de type Works-flow (enchaînement de blocs fonctionnels dont les entrées n'ont pas d'influence sur les sorties des blocs précédents). Les modèles créés avec VHDL-AMS peuvent donc aussi bien être descriptifs que prédictifs. Deux autres caractéristiques de VHDL-AMS, dont chacune est partagée avec quelques autres langages, sont le traitement des équations implicites (c'est-à-dire des équations où l'inconnue ne se trouve pas forcément dans le membre de gauche) et à travers celles-ci, l'utilisation des lois de Kirchhoff généralisées, fondement des relations implicites entre les différents nœuds d'un système.

I.4.4 Limite de VHDL-AMS :

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

extensions et des modifications d'anciens simulateurs, et pas encore sur de nouvelles techniques de simulation spécifiques, implique des limitations dans les possibilités de simulation qui empêchent l'implémentation de certaines instructions du langage. C'est le cas de l'instruction PROCEDURAL qui pourrait nous permettre de simuler un bloc d'instructions séquentielles à chaque ASP par exemple. La méthode de résolution matricielle des simulateurs empêche quant à elle d'implémenter une instruction comme DISCONNECT qui permettrait de retirer un élément du système, ce qui modifierait dans le même temps la structure de la matrice de calcul. Or cette opération ne peut être refaite en cours de simulation [9].

I.4.5 Evolution liée à VHDL-AMS :

Avec la réaffirmation de la normalisation VHDL-AMS prévue en 2004, et son évolution future, une partie des problèmes présentés ci-dessus devrait trouver une solution. Par ailleurs, le langage verra ses possibilités prochainement étendues grâce à l'extension Radio Fréquence/Micro Ondes (Radio Fréquence / Micro Waes - RF/MW) en cours de développement. Des pistes complémentaires sont également étudiées pour élever le niveau d'abstraction des connexions, afin d'autoriser l'utilisation de différentes classes de signaux sur un même dispositif de connexion en fonction du niveau d'abstraction du modèle. Il est également question d'incorporer des primitives SPICE à VHDL-AMS et d'y adjoindre des mécanismes automatiques de spécification et de vérification des modèles. Par ailleurs, avec les possibilités grandissantes offertes par la nouvelle génération de simulateurs VHDL-AMS, présentée ci-après, il est légitime de se demander si les efforts de recherche pour développer des simulateurs spécialement dédiés à la simulation analogique supportant le jeu complet d'instructions VHDL-AMS seront encouragés. En effet, des logiciels comme S'implorer permettent de pallier à beaucoup de problèmes, comme l'absence du PROCEDURAL, par des moyens détournés comme l'utilisation de modèles C/C++ ou celle de composants utilisant le "métalangage" SML de S'implorer qui peut communiquer avec le noyau de simulation VHDL-AMS et réagir sur ses ASP. Ceci a cependant l'inconvénient d'ôter la portabilité du système et il est peu probable qu'un palliatif puisse être trouvé aux problèmes du type DISCONNECT qui relèvent directement du fonctionnement du noyau de simulation. Les concepteurs font sans ces instructions pour l'instant, devront-ils continuer ainsi, avec à terme l'abandon de celles-ci dans la norme, ou bien les noyaux de simulation VHDL-AMS vont-ils évoluer

I.4.6 Choix du logiciel de simulation :

Comme énoncé précédemment, notre choix s'est porté sur le VHDL-AMS, en partie à cause de la diversité des simulateurs disponibles pour ce langage. En effet, l'offre logicielle dans ce

domaine est très importante, mais également très variée. Outre la concurrence qui fait naître des outils similaires, l'évolution dans l'utilisation du langage lui-même a fait émerger plusieurs philosophies de conception. Nous avons donc dû choisir parmi tout cela le simulateur le plus adapté à nos besoins.

I.4.7 Possibilité offerte par l'interfaçage avec VHDL-AMS :

Important de pouvoir concevoir des modèles qui fassent cohabiter des composants de différents langages. Pour cela, il faut pouvoir les interfacier. La plupart des logiciels du marché permettent ces interactions, mais elles sont le fruit de démarches propriétaires. En effet, la norme VHDL-AMS ne donne aucune information sur la façon dont le langage devrait interagir avec d'autres. De ce fait, la portabilité des modèles complexes est extrêmement réduite. Il serait intéressant que la norme définisse une façon d'interagir avec l'extérieur (un prototype d'appel de fonction ou une instruction propre par exemple), de manière à donner une direction aux développeurs de simulateurs. C'est cependant une tâche très ardue si l'on veut essayer de mettre au point un mécanisme simple qui permette d'accéder à n'importe quel langage étranger sans pour autant en limiter les possibilités [9].

I.5 Utilisation des modèles VHDL-AMS par un simulateur

L'utilisation des modèles VHDL-AMS par un simulateur passe par trois étapes principales : la compilation, l'élaboration et la simulation. Il est important de comprendre ce qui différencie les opérations effectuées par les étapes de compilation et d'élaboration. La réutilisation par le support de la généricité est largement favorisée par la séparation de ces deux opérations.

I.5.1 Compilation :

- Analyse syntaxique et sémantique.
- Unité de conception primaire avant l'unité de conception secondaire.
- Stockage en bibliothèque si réussite de la compilation.

I.5.2 Elaboration :

- Rassemble les modèles utilisés (propriétaires ou IP (intellectual property)) définis dans la configuration.
- Vérifie les associations, fixe les génériques, crée les data-structures.
- Interconnectés les processus et faire les jeux d'équations simultanées.
- Initialise les horloges.

- Initialise les objets.

I.5.3 Simulation :

La plupart du temps deux noyaux de simulation sont utilisée afin de :

- Vérifier les contraintes dynamiques de valeur.
- Rapporter les REPORT et piloter le simulateur avec le SEVERITY.
- Vérifier les boucles infinies des processus, contrôler la convergence.
- Tenir compte des BREAK.

I.5.4 Exploitation :

- Tracé des chronogrammes, tracé des x(y).
- Auto - test des modèles.

Remarque : 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 contient autant d'équations que d'inconnues. Le concepteur doit alors s'assurer que le modèle remplit le critère de solvabilité imposé par la norme. Le nombre d'équations simultanées (SS) doit être égal au nombre de quantités THROUGH (QT) augmenté du nombre de quantités FREE (QF) et du nombre de quantités d'interface en mode OUT (Q_{out}).

$$SS = QT + QF + Q_{out}$$

Attention :

Ce critère n'assure pas la convergence qui est une caractéristique dynamique du réseau d'équations couplés aux techniques d'analyse numériques mises en place. En théorie ce critère est suffisant mais pas localement nécessaire, il est de ce fait assez contraignant [6].

I.6 Aperçu de la modélisation VHDL-AMS en analogique :

Le langage VHDL-AMS est donc un langage inspiré de VHDL pour permettre l'extension à l'analogique et au mixte. Les algorithmes de simulation ne font pas partie du langage. On pourra alors décrire des modèles mixtes mais aussi multi-technologiques (thermiques, mécaniques, hydrauliques, etc...). Dans ce contexte, l'objectif de notre travail est alors de permettre rapidement de développer des modèles sous VHDL-AMS. Ceci nécessite donc une certaine

connaissance du langage. Cette partie décrit ce qui nous semble fondamental pour aborder la modélisation VHDL-AMS dans le domaine analogique.

I.6.1 Structure d'un modèle VHDL-AMS :

ENTITY et ARCHITECTURE :

La modélisation en VHDL-AMS de tout composant pris au sens large du terme, se fait au moyen de deux types d'objets : l'ENTITY et l'ARCHITECTURE.

- l'ENTITY décrit la vue extérieure du composant. Nous pouvons comparer l'ENTITY a une boîte noire où seules les entrées-sorties du composant sont visibles. Lors de l'écriture d'une ENTITY, nous ne déclarons 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
```

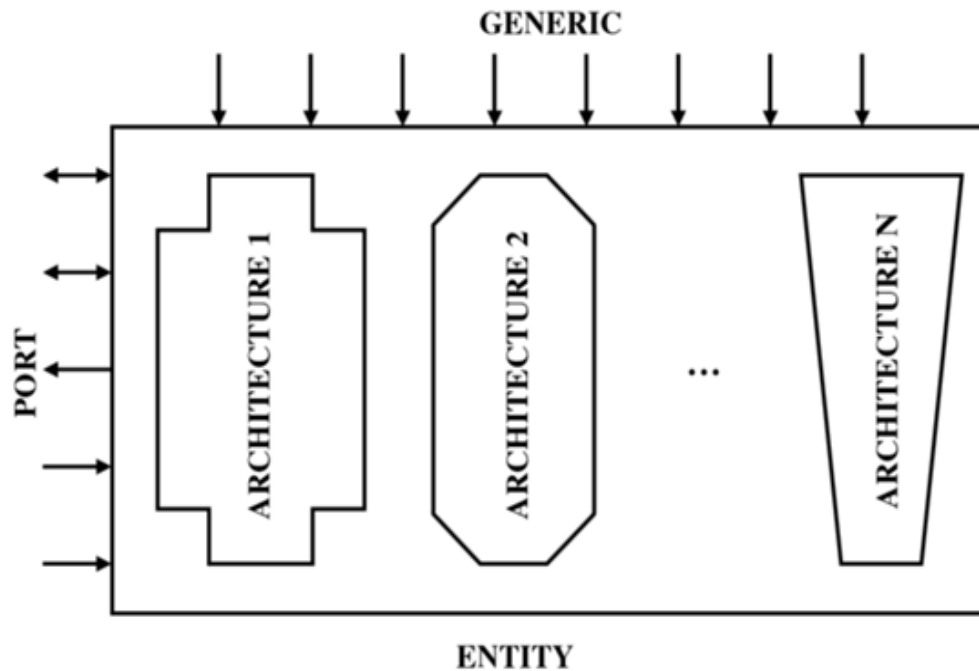
- l'ARCHITECTURE représente une des descriptions possibles de la fonction du modèle. Une ARCHITECTURE se réfère toujours à une unique ENTITY et contient la description de la fonction réalisée par l'ENTITY. Ainsi, si la boîte noire est l'ENTITY, c'est l'ARCHITECTURE qui va dicter le comportement de cette boîte noire. Pour une ENTITY donnée réalisant une fonction précise, il peut y avoir autant d'ARCHITECTURE que de manières de décrire la fonction à réaliser. L'ARCHITECTURE contient donc les équations de la fonction et les déclarations de toutes les variables dites locales, c'est-à-dire qui n'ont pas de raison d'exister hors de l'ARCHITECTURE.

Exemple de syntaxe :

```
ARCHITECTURE nom_de_l'architecture OF nom_de_l'entité  
déclaration_des_variables
```

BEGIN

Déclaration_des_équations

END [ARCHITECTURE] nom_de_l'architecture**Figure I.2** : structure d'un modèle VHDL

Il faut bien comprendre ici que l'on peut avoir autant d'architectures réalisant une fonction (celle de l'entité) que l'on peut trouver de manières de décrire cette fonction. Une fois le concept d'ENTITY et d'ARCHITECTURE assimilé, il faut maintenant se pencher sur les moyens mis à disposition par VHDL-AMS pour remplir et faire fonctionner notre boîte noire.

I.6.2 Les nouveaux objets par rapport au VHDL : QUANTITY et TERMINAL :**a) Origine de la QUANTITY :**

Le VHDL-AMS sert à décrire des systèmes mixtes continus/discrets. Le caractère continu de ces systèmes est décrit par des **EDA** (**E**quations ordinaires **D**ifférentielles **A**lgébriques) avec le temps comme variable indépendante. Ces EDA_s peuvent s'écrire sous la forme :

$$F(x, dx/dt, t) = 0 \quad \text{vecteur d'expressions}$$

x : vecteur d'inconnues

t : temps

Les solutions analogiques des EDA_s n'obéissent pas à la même méthode de résolution que le VHDL-AMS. Celui-ci introduit alors une nouvelle classe d'objets porteurs de valeurs : la QUANTITY, qui va représenter les inconnues des EDA_s dans le domaine analogique. Dans le paragraphe suivant nous allons voir quelques propriétés et règles de syntaxe s'appliquant aux QUANTITY.

b) Définition de la QUANTITY :

La QUANTITY est donc un objet, dont la valeur à temps précis est solution des EDA_s . Pour ce faire, les QUANTITY doivent avoir des sous-éléments scalaires de type virgule flottante pour approximer les nombres réels solutions des EDA_s . Un objet QUANTITY peut se trouver dans une expression partout où une valeur de ce type est permise. Il existe des QUANTITY scalaires et d'autres composites. Les caractéristiques d'une QUANTITY composite sont simplement l'agrégation des caractéristiques des sous-éléments scalaires.

Syntaxe : **QUANTITY** identifier_liste : real; -- la déclaration de deux quantités dites libres

Exemple : **QUANTITY** q1, q2 : real;

Syntaxe : **QUANTITY** identifier_liste : real := flot; -- la déclaration d'une quantité dite libre

Exemple : **QUANTITY** q1 : real := 1.0;

Dans ce cas l'expression représente une valeur initiale de la quantité déclarée. En l'absence de la valeur initiale, une valeur par défaut est appliquée. Chaque quantité de type scalaire et chaque sous-élément scalaire d'une quantité composite est une quantité scalaire.

b.1) PORT (QUANTITY) :

Mais les QUANTITYs peuvent aussi être déclarées comme éléments d'interface dans une liste de PORT. Et à ce moment là, il s'appelle un PORT QUANTITY (terme analogue à celui employé en VHDL, le port signal). Exemple de syntaxe :

ENTITY exemple **IS**

PORT (**QUANTITY** x1 : IN real);

PORT (**QUANTITY** x2 : OUT real);

END ENTITY exemple;

b.2) QUANTITY implicite :

Pour chaque quantité « Q » déclarée nous avons accès à des quantités implicites telles que :

- Q'Dot : la dérivée de « Q » en fonction du temps.
- Q'Integ : l'intégrale de temps de la quantité « Q » du temps zéro au temps présent.

b.3) QUANTITY ACROSS, THROUGH, et TERMINAL:

Un langage de type **VHDL-AMS** permet de décrire des modèles comportementaux. On tire de cette opportunité des avantages qui permettent d'étendre la notion d'analogie à la notion de multi-technologie. Les domaines technologiques pouvant être appréhendés sont très variés. Un domaine peut être caractérisé/spécifié par les deux objets **VHDL-AMS** suivants : **ACROSS** et **THROUGH**. L'analogie avec le domaine électrique permet de comprendre aisément leur signification. En effet, dans le domaine électrique **ACROSS** est la tension aux bornes d'une branche et **THROUGH** est le courant circulant entre les bornes de cette branche. Le tableau I.2 présente une liste (non exhaustive) de domaines technologiques et de leurs caractéristiques. Certains domaines peuvent paraître inattendus, comme le domaine financier. Avec **VHDL-AMS**, on a la possibilité de créer d'autres domaines à condition de trouver leurs objets caractéristiques **ACROSS** et **THROUGH** associés, ainsi que les équations spécifiant la relation entre ces deux objets et représentant le comportement du modèle (tableau I.1).

DOMAINE	ACROSS	THROUGH
Electrique	Tension (V)	Courant (A)
Thermique	Température (°C)	Puissance (W)
Mécanique linéaire	Position (m)	Force (N)
Mécanique rotative	Vitesse angulaire (radian/s)	Moment (N*m)
Magnétisme Force	magnétomotrice (n*A)	Flux (Wb)
Hydraulique	Pression (Pa)	Débit (l/s)
Financier	Dette (DA*mois)	Monnaie (DA)
Radiatif	Dose (rad)	(Photo) courant (A)

Tableau I.1: Quelques domaines d'application.

Les Quantités de branchement représentent les inconnues dans les équations qui décrivent les systèmes conservatifs. On a deux types de **QUANTITY** de branchement :

- le type **ACROSS** (aux bornes, grandeur d'effort).
- le type **THROUGH** (à travers, grandeur de flux).

Les types ACROSS représentent les effets de « type potentiel » comme la tension dans le domaine électrique, la pression dans le domaine hydraulique. Les types THROUGH représentent le courant dans le domaine électrique, le flux dans le domaine magnétique. Une QUANTITY de branchement est déclarée par référence à deux TERMINAL. Le TERMINAL est le deuxième nouvel objet introduit par le VHDL-AMS. Un TERMINAL est déclaré comme étant de nature simple ou composite dont les sous-éléments sont de nature simple. Chaque nature représente une discipline physique : électrique, thermique, etc....

Example :

SUBTYPE voltage **IS** real; -- ici “voltage” et “current” sont déclarés

SUBTYPE current **IS** real; come sous-type, “electrical” come

NATURE electrical IS nature.

Voltage **ACROSS**;

Current **THROUGH**;

TERMINAL t1, t2, electrical; -- t1 et t2 sont les TERMINALs.

QUANTITY v across i1, i2, through t1 to t2;

-- v est la tension entre ces deux

TERMINALS et i_1, i_2 sont deux

Branches parallèles représenta le courant.

Nous venons de voir que la déclaration des QUANTITYs de branchements se fait par la déclaration de deux TERMINALs t1 et t2 avec t1 en TERMINAL positif et t2 en TERMINAL négatif. La direction d'une branche va de + vers -, direction du courant positif. Un TERMINAL peut être déclaré partout où un signal est déclaré en VHDL. Il peut aussi être un élément d'interface comme nous l'avons vu avec les quantités. Dans ces conditions, on parle de PORT TERMINAL.

Exemple :

PORT (TERMINAL anode, cathode : electrical); L'association de PORT TERMINAL est utilisée dans la construction de nœuds dans des descriptions hiérarchisées, comme le PORT SIGNAL en VHDL.

I.6.3 Déclaration implicite :

La déclaration d'une nature N crée un TERMINAL de référence pour tous les TERMINALs de cette même nature. Le TERMINAL de référence de la nature N est noté N'référence. Par exemple pour la nature électrique, les TERMINALs électriques sont la masse comme TERMINAL de référence. La création d'un TERMINAL T engendre deux QUANTITYs implicites :

- La QUANTITY de référence T'référence, qui est un ACROSS avec T et N'référence en TERMINALs + et –
- La QUANTITY de contribution T'contribution, qui est un THROUGH de valeur égale à toutes les QUANTITY THROUGH incidentes à

T (avec le signe qui convient). Si T apparaît comme effectif dans l'association de PORT alors les quantités de contribution correspondantes sont à ajouter à la somme.

I.7 Conclusion :

Dans ce chapitre, nous avons présenté une initiation détaillée au langage VHDL-AMS. Une méthode de modélisation par le biais de ce langage sera présentée dans le chapitre suivant en considérant quelques composants élémentaires tels que la résistance, l'inductance, la capacité, et la diode.