LEXIQUE

AFWAL	Air Force Wright Aeronautical Laboratories
ASIC	Application Specific Integrated Circuit
ASP	Analog Simulation Point
CLSI	CAD Langage Systems Inc.
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
LSP	Logic Simulation Point
MRT	Minimum Resolvable Time
PLL	Phase Locked Loop – Boucle à verrouillage de phase
Soc	System on Chip
SPICE	Simulation Program for Integrated Circuits Emphasis
UMTS	Universal Mobile Telecommunication System
VASG	VHDL Analysis and Standardization Group
VCO	Oscillateur Contrôlé en Tension
VHDL	Very High Speed Integrated Circuits Hardware Description Language
VHDL-AMS	Very High Speed Integrated Circuits Hardware Description Language for Analog and Mixed Signal

LISTE DES TABLEAUX

LISTE DES FIGURES

SOMMAIRE

INTRODUCTION

Les progrès accomplis en VLSI ont permis de combiner sur un même système les deux sous-systèmes analogiques et numériques. Les concepteurs de circuits électroniques intègre des System on Chip (SOC) et des ASIC (Application Specific Integrated Circuit) mixtes c'est-à-dire intégrant des fonctions analogiques et numériques. Les simulateurs sont devenu les outils principales dans la phase de conception, ceci dans l'objectifs de minimiser le coût et le temps de la conception et afin d'obtenir des circuits répondant aux spécifications des cahier des charges.

Ces nouvelles tendances font que les simulateurs traditionnels sont limités en performances, un circuit complexes nécessite un temps de simulation très important, de plus le système doit être soit à temps discret pour le cas d'un système numérique, soit à temps continu pour un système analogique, soit les deux en même temps, et ce comportement doit aussi être compréhensible par le simulateur.

Les dernières générations de simulateurs : les simulateurs mixtes analogique numérique sont apparus avec le développement des langages de description matérielle tel VHDL, qui présente les avantages de supporter la description de systèmes électronique à la fois numériques et analogiques mais aussi d'autres systèmes tels que l'électromécaniques, thermique, hydraulique,etc.…. Et aussi de renforcer la cohérence des outils logiciels utilisés pour la simulation et la synthèse, de supporter plusieurs niveaux d'abstraction et autorise des descriptions hiérarchiques

Pour palier aux problèmes de temps de simulation, la première solution proposée était de remplacer le circuit par des modèles équivalents reproduisant le plus fidèlement possible les performances du circuit : c'est l'objectif de la modélisation. La technique de modélisation comportementale est apparue avec l'avènement du langage de description matérielle. Elle aide à la résolution des problèmes de convergences mais aussi elle permet de mettre en œuvre la méthodologie de conception hiérarchique pour réaliser un circuit répondant au premier coup aux spécifications. Elle est ainsi devenue indispensable pour la validation des systèmes complexes

Les travaux présentés dans ce mémoire intitulé « Modélisation comportementale par VHDL-AMS d'un synthétiseur de fréquence » ont pour objectif principal d'introduire la modélisation comportementale de systèmes mixtes à l'aide du langage de description matérielle VHDL-AMS. Des bibliothèques de modèles pour le domaine de la télécommunication ont été élaborées.

Ce mémoire est présenté comme suit :

Le premier chapitre décrit les différentes techniques nécessaires dans la conception d'un système : la simulation, les méthodes de description et de conception. Ceci a pour objectif d'introduire l'importance de la modélisation comportementale ainsi que des outils choisis dans la réalisation de ce mémoire. Le chapitre 2 présente les éléments essentiels du langage VHDL-AMS. La chapitre 3 s'intéresse à la modélisation comportementale : les méthodes utilisés, les environnements de travail. Deux approches de modélisation seront traités : schématique et fonctionnelle. L'approche fonctionnelle développée au chapitre 3 a été utilisée pour développer les modèles comportementaux pour la synthèse de fréquence dans le chapitre 4. Les détails sur la détermination de tous les paramètres du modèle sont présentés dans l'Annexe 3. L'outil SYSTEM VISION de Mentor Graphics a été utilisé pour tester les modèles. Un guide d'utilisation de cet outil est fourni dans l'Annexe 1.

Chapitre 1: Introduction à la conception de système

I- Introduction

Les simulateurs sont les outils essentiels d'aide à la conception et validation d'un système électronique. Ces simulateurs sont basés autour de quatre ensembles de données et programmes [1] :

- Le moyen de décrire le système à simuler : langage de description ;
- La description du système : modèle ;
- La description des interfaces du système avec l'extérieure : entrée et sortie ;
- Le mécanisme de simulation du système : simulateur.

II- La simulation

La simulation est essentielle dans la conception de circuits en tant qu'outil de validation des choix du concepteur. Pour des raisons de compétitivité, elle doit être la plus rapide et la plus fiable possible. Suivant le type du système, on distingue généralement trois catégories de simulateurs:

- les simulateurs analogiques ;
- les simulateurs numériques ;
- et les simulateurs mixtes analogique numérique.

2.1-La simulation analogique

Elle traite des signaux continus dans le temps et est utilisée pour déterminer les performances électriques des circuits. On l'appelle aussi simulation électrique. La référence en matière de simulateur analogique de circuits intégrés est le programme SPICE, développé à l'université de Berkeley. Il existe actuellement de nombreuses versions industrielles de ce programme basées sur le même langage de description structurelle SPICE.

La première étape de la simulation analogique consiste en la mise en équation du réseau électrique par application des lois de Kirchhoff. La taille du système d'équations est une fonction exponentielle du nombre de noeuds et conditionne fortement la vitesse de simulation.

Le simulateur procède ensuite en la résolution d'équations différentielles et algébriques linéaires ou non linéaires. Les solutions sont des tensions entre les nœuds du circuit et les courants entre les branches du circuit.

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

- **Analyse DC :** étude du point de fonctionnement du circuit qui correspond à un régime permanent.
- **Analyse temporelle :** étude de la réponse temporelle du circuit.
- **Analyse AC :** étude de la réponse fréquentielle ou petits signaux, pour laquelle le circuit est linéarisé autour du point de fonctionnement. Les analyses de bruits, la définition des pôles et zéros peuvent être aussi effectuées à l'issue d'une analyse AC.
- **Analyse statistiques :** détermination de la dispersion des performances du circuit en fonction des fluctuations statistiques de paramètres de conception. Cette étude permet ensuite de définir la valeur nominale des composants pour obtenir un rendement optimal. Un grand nombre de simulations sont ici requises.

2.2-*La simulation numérique*

Elle manipule des signaux discrets et quantifiés (0,1, indéterminé ou X,..) et se caractérise par une très grande rapidité.

La simulation est basée sur l'exécution conditionnelle et itérative d'équations logiques dépendantes dans un temps discrétisé. Ainsi un simulateur numérique doit avoir une notion de temps c'est-à-dire maintenir un compteur de temps, le temps physique courant, et attribuer une date physique à chaque évènement au sein de la simulation. [2]

Le pas de simulation n'a pas de valeur temporelle physique intrinsèque. C'est un intervalle de temps virtuel ou symbolique, appelé souvent delta, dont la durée est nulle et qui ne sert qu'à ordonner les évènements simultanés. Pendant un delta, le temps physique ne s'écoule pas.

La simulation procède par pas :

- soit en incrémentant le temps symbolique (*delta*) jusqu'à ce que l'état du circuit se stabilise. Les évènements traités sont alors simultanés d'un point de vue temporel physique
- soit en sautant directement à la date physique du prochain évènement prévu, si plus aucun évènement n'est prévu pour la date physique actuelle.

Les affectations de variables doivent être instantanées si les variables sont locales à un processus (programme séquentiel), et différés à la fin du delta courant si ces variables sont des signaux de communication entre processus. Chaque processus est une boucle infinie qui doit être stoppée par un point d'arrêt implicite ou explicite, sinon le temps physique ne s'écoule pas. Ce point d'arrêt définit une liste de sensibilité. Un processus n'est alors exécuté (réveillé) que lors d'un évènement portant sur un signal membre de cette liste de sensibilité.

Le tableau 1 récapitule les caractéristiques principales de la simulation numérique et de la simulation analogique.

Tableau 1 : Récapitulation des caractéristiques de la simulation numérique et analogique [3]

Caractéristiques	Simulation numérique	Simulation analogique
Variables/Inconnues	Signaux logiques	Tensions, courants,etc…
Valeurs des inconnues	Quantifiées (‘0’,’1’,’X’,’Z’,etc,…)	Réelles
Calcul de l’état du circuit/modèle	Evaluation de fonctions logiques	Résolutions d’équations différentielles algébriques non linéaires
Etat initial (t=0)	Pas nécessairement un état stable	Etat stable (point de repos DC) requis
Itérations à un temps donné	Affectation de signaux avec délais nuls (délai delta)	Résolution de systèmes non linéaires
Représentation du temps	Discret, Multiple du MRT	Réel
Gestion du temps	Dirigée par évènements	Continue avec pas d’intégration variable
Contrôle du pas temporel	Evènements sur les signaux	Erreur de troncature locale ou équivalente
Types d’analyse	Temporelle	Temporelle, DC, AC

2.3-*La simulation analogique numérique*

La simulation mixte analogique numérique permet d’étudier le comportement temporel de systèmes complexes. Au vu du Tableau 1, la simulation mixte analogique numérique doit procéder suivant trois phases :

- **la phase d’élaboration** : les blocs analogiques et numériques constituant le circuit mixte est partitionné dans cette phase. Chaque partie sera traitée par les algorithmes correspondants.
  La correspondance des données entre les algorithmes analogiques et digitaux est assurée par des modèles plus ou moins élaborés de convertisseurs A/D et D/A placés entre les deux parties.
- **la phase d’initialisation :** elle correspond à la détermination de l’état initial des grandeurs mises en jeu (tensions, courants, états logiques). Ceci correspond à une analyse DC et est indispensable pour le simulateur analogique. Pour le simulateur numérique, cela peut correspondre soit à une initialisation (solution au temps 0), soit au temps au bout duquel un état stable est trouvé.
- et **la phase de simulation :** résolution des problèmes de synchronisation des algorithmes électriques et numériques, qui ont des gestions différentes du pas de temps.

III- Les différents types de langage de description

Nous pouvons distinguer principalement deux grandes familles de langages :

- les langages de description logicielle ;
- les langages de description matérielle.

3.1-Les langages de description logicielle

Souvent appelé langage de bas niveau, les langages de description logicielle tel que C, Fortran, C++, sont surtout utilisées pour coder les primitives des simulateurs électriques.

Dans certains cas, le modèle décrit à l'aide d'un langage de description matérielle est traduit en langage de bas niveau (exemple : HDL traduit en C) avant sa compilation. Certains simulateurs (exemple : Eldo) offrent aussi des bibliothèques de fonctions permettant à l'utilisateur d'écrire ses propres modèles analogiques en C. Le code compilé des nouveaux modèles doit être archivé dans une bibliothèque qui sera liée au simulateur à l'exécution.

3.2-*Les langages de description matérielle:*

Un langage de description matérielle est un outil de description, éventuellement formel, du comportement et de la structure d'un système matériel.

Suivant la description envisagée ont peut distingué deux types de langage de description:

3.2.1- Le langage de description structurelle

La description structurelle donne des informations sur la structure des blocs et composants utilisés. Le plus connu est le langage d'entrée du simulateur SPICE nommé SPICE. Il permet de décrire le réseau électrique du circuit pour être analysé par le simulateur afin de construire un système d'équations, basés sur les équations de Kirchhoff et les équations des composants.

Il peut être aussi utilisé pour la description comportementale de fonctions analogiques. Ce type de langage présente l'avantage d'être simple. Son utilisation ne nécessite pas l'apprentissage d'un langage de programmation mais requiert simplement une bonne connaissance du simulateur.

Les applications de ce langage de description sont liées aux caractéristiques du simulateur associé et à la formulation des équations du réseau. Pour le cas de SPICE elle est limitée à la description des circuits analogiques.

3.2.2- Le langage de description comportementale

Un langage de description comportementale est distinct d'un langage de programmation classique dans la mesure où il manipule de nouveaux types de données selon des lois adaptées à la description physique des composants. En analogique comme dans les autres domaines physiques, on applique les lois de Kirchhoff et les lois de conservation de l'énergie.

Le standard dans le domaine numérique est le langage VHDL. Le langage VHLD-AMS définit les extensions analogiques du standard VHDL et permet aussi la description de systèmes mixtes analogiques numériques pouvant appartenir à différents domaines physiques : systèmes électriques, mécaniques, thermiques, etc.….

Un langage de description matérielle comme VHDL-AMS présente les caractéristiques suivantes :

- supporte la description de systèmes à la fois logiques et analogiques ;
- permet la description de l'état de la conception pour toutes les étapes du processus ;
- renforce la cohérence des outils logiciels utilisés pour la simulation et la synthèse ;
- indépendant de toute méthodologie de conception, de toute technologie de fabrication et de tout outil logiciel ;
- supporte plusieurs niveaux d'abstraction et autorise des descriptions hiérarchiques ;
- Standardisé par l'intermédiaire d'organisations reconnues comme IEEE, ANSI ou ISO.

IV- La description du système : modèle

4.1*Les différents niveaux de description*

Il existe plusieurs niveaux d'abstraction pour la description d'un système et ces niveaux sont aussi différents pour le domaine numérique et analogique.

Chaque niveau est caractérisé par les deux types de représentation suivants:

- **la représentation comportementale** qui est à un niveau assez abstrait et qui est indépendante de toute architecture ;
- et **la représentation structurelle** qui décrit une architecture donnée à l'aide d'éléments appartenant au niveau inférieur.

4.1.1 Domaine numérique

Le tableau 2 suivant présente les différents niveaux d'abstraction pour la description des systèmes numériques.

Tableau 2 : Niveaux d'abstraction de systèmes numériques [4]

Niveaux d'abstraction	Représentation comportementale	Représentation structurelle
Système	Schémas synoptiques,	Processeurs, Mémoires

	Algorithmes	
Microarchitecture	Registre de transfert (RTL)	Registres, ALUs
Logique	Equations booléennes, diagrammes d'états	Ports logiques
Circuit	Fonctions de transfert, diagrammes temporels	Transistors interconnectés

4.1.2 Domaine analogique

Par analogie à la hiérarchie présentée par le tableau 2 dans le domaine numérique, le tableau 3 ci-dessous présente les différents niveaux d'abstraction pour la description des systèmes analogiques.

Tableau 3 : Niveaux d'abstraction de systèmes analogiques [5]

Niveaux d'abstraction	Représentation comportementale	Représentation structurelle
Système	Fonctions de transfert Schémas blocs H(s), H(z) (Domaine fréquentiel, Domaine temporel Domaine analogique/digital)	Convertisseurs A/D,D/A PLL Filtres Sommateur, intégrateur, multiplieur,…
Fonctionnel	Equations algébriques linéaires et non-linéaires Courbes de transfert Tables	Amplificateur opérationnel Sources de tension ou de courant Comparateur
Circuit	Macro-modèles	Transistor Eléments passifs R, L, C Et les autres composants actifs
Composant	Modèles de composants	Layout des composants

4.2 *Modélisation d'un système*

La modélisation a pour but de caractériser par une fonction mathématique ou un modèle numérique les différents composants qui constituent le circuit ou le système. Cette partie est très délicate puisque la précision du système dépend du modèle élaboré. Le plus important critère de la modélisation est que le modèle doit être le plus fidèle possible et le plus exact possible.

Le modèle d'un système est une représentation de son comportement à l'aide de laquelle le simulateur comprend et procède à des calculs. Suivant le type de langage utilisé pour la modélisation on peut distinguer deux types de modélisation :

- **modélisation structurelle :** qui consiste à décrire la structure d'un système en décrivant les interconnexions entre éléments ;

- **modélisation comportementale :** qui consiste à modéliser le système ou le circuit par l'évolution de ses sorties en fonctions des entrées en réponse à différents stimuli.

4.2.1 La modélisation structurelle

Cette méthode consiste :

- en la construction d'un schéma qui conduira aux relations souhaitées entre des variables représentées par les tensions de nœud et les courants de branche ;
- ou en la simplification d'un schéma afin de réduire le nombre de nœuds du circuit initial. On définit des blocs fonctionnels à partir d'expressions mathématiques et de fonctions.

On utilise pour cela un langage de description structurelle tel SPICE, cette méthode est souvent appelée : macro-modélisation. Les macro modèles sont construits à partir d'un nombre réduit de composants primitifs du simulateur. On y inclut des éléments passifs comme les résistances, capacités ou autres, de sources dépendante et indépendante de type courant ou tension.

En grande partie cette méthode s'est développée grâce au succès du simulateur SPICE et aux besoins des concepteurs de faire apparaître des phénomènes autres que ceux électriques (modélisation de phénomènes physiques par schéma électrique équivalent) ou de simplifier un schéma en remplaçant certaine parties par des fonctions plus simples (amplificateur opérationnel, etc.…).

4.2.2 Modélisation comportementale

La description comportementale exprime le fonctionnement du bloc à l'aide des équations sans se soucier de sa structure interne. Contrairement à la macro modélisation qui utilise les primitives disponibles du simulateur, la modélisation comportementale consiste en quelque sorte à créer de nouvelle primitive.

Cette méthode permet de concevoir des circuits de plus grande qualité car la description comportementale de chaque bloc du circuit conduit à une définition très précise de ses spécifications, ce qui permet d'éviter des erreurs de conception et d'obtenir un circuit optimal.

V- Méthodologies de conception

5.1- *Objectifs d'une méthode de conception*

Une méthode de conception est définie par les étapes que le concepteur décide de suivre depuis le cahier des charges jusqu'au layout. Les objectifs principales étant :

- la sûreté de conception c'est à dire l'obtention d'un circuit correct. Pour cela il leur faut détecter rapidement les erreurs de fonctionnement avant même d'établir une description niveau transistor ;
- la réduction du temps nécessaire à la conception en résolvant le problème de temps de simulation important pour les systèmes électroniques qui sont de plus en plus complexes ;
- et la réduction des coûts.

5.2- *Les différentes méthodes de conception*

5.2.1 Méthode descendante (*Top Down*)

Cette méthode est actuellement très utilisée afin d'aborder la conception des systèmes qui sont devenu de plus en plus complexe.

Pour cette approche, on part d'une description fonctionnelle du système et on décompose progressivement son architecture jusqu'au niveau transistor. Après une spécification du système à concevoir, on vérifie sa fonctionnalité avec une description fonctionnelle, donc à un haut niveau d'abstraction. On peut imaginer après plusieurs niveaux de description fonctionnelle qui vont permettre de vérifier les différentes fonctions des sous blocs construisant le système global. On passe ensuite à la synthèse pour obtenir une description schématique an niveau élémentaire (portes logiques ou niveau transistor). A partir de cette description, on peut générer le layout à l'aide d'outils de routage.

La figure1.1 présente les différentes étapes suivies dans une approche *Top-Down*.

Cette approche permet de vérifier le bon fonctionnement du système avant de passer à une description niveau transistor et de détecter des erreurs de conception précoces. Elle permet également de reporter le choix de la technologie le plus tard possible dans le cycle de conception.

5.2.2 Méthode ascendante (*Bottom-Up*)

La description traditionnelle des circuits et systèmes analogiques s'appuie sur des composants de base (transistor, diode, résistance, etc.…). Cette méthode de conception se base donc sur une description au premier niveau (niveau transistor). Les transistors sont d'abord assemblés pour créer une fonction, laquelle est utilisée dans un bloc regroupant plusieurs fonctions et ainsi de suite.

La figure1.1 présente les différentes étapes suivies dans une approche *Bottom-Up*.

Avec l'apparition des langages de description matérielle, la méthode ascendante ne se limite plus au premier niveau élémentaire. Des modèles comportementaux des blocs constituant le système peuvent être extraits de leur description schématique. On peut encore remonter dans les niveaux d'abstraction pour passer de la vérification fonctionnelle des blocs de tout le système.

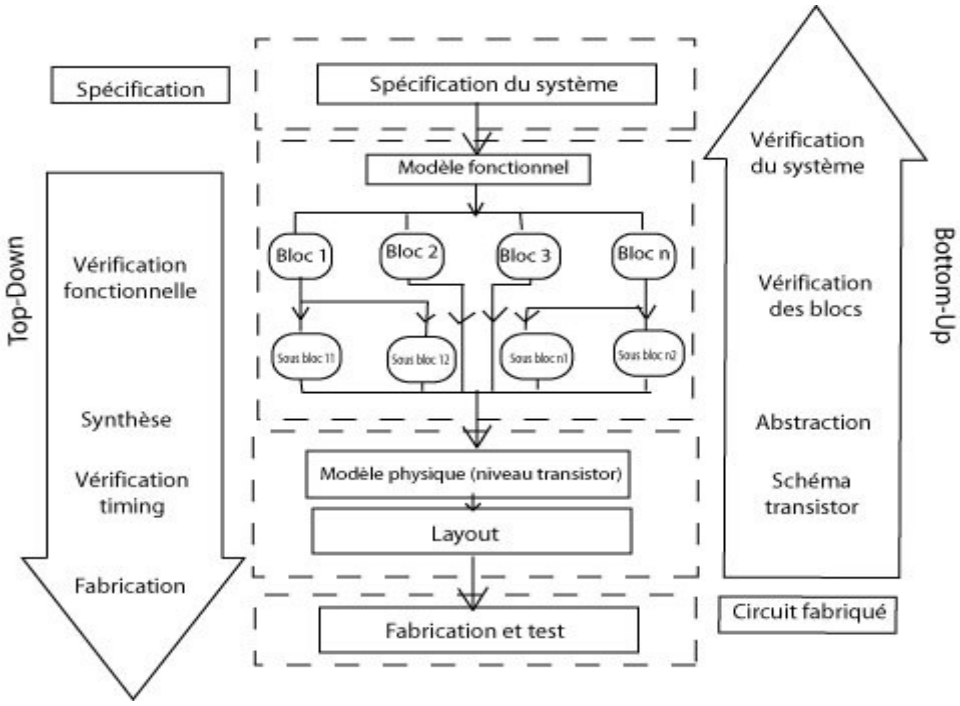


Figure 1.1 : Méthodologie de conception hiérarchique *Top Down* et *Bottom Up* [5]

En réalité, les concepteurs utilisent un mélange des deux approches ascendante et descendante. Par exemple on peut imaginer une conception *Top Down* qui utilise des modèles de base issus de l'approche *Bottom Up*.

VI- Conclusion

Pour accélérer le cycle de conception, il est souvent d'usage de reprendre des blocs déjà conçus par un autre concepteur ou un autre organisme. La modélisation comportementale prend ici tout son sens car le modèle peut devenir la carte d'identité d'un circuit sans que l'on connaisse son architecture. L'adaptation de la méthodologie hiérarchique de conception *Top Down* et *Bottom Up* a permis l'automatisation de conception, et ceci aussi grâce à l'apparition des langages de description matérielle tel le VHDL-AMS qui a largement aidé l'avènement de la modélisation comportementale. Ce langage sera présenté dans le chapitre suivant.

Chapitre 2: Présentation du langage VHDL-AMS

I- Introduction

Le langage VHDL est un standard IEEE (IEEE 1076-1993) pour la modélisation, la simulation et la synthèse de systèmes matériels logiques. 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. [3]

VHDL a été développé par le Groupe d'Analyse et de Standardisation VHDL. *Saunders* est le coordinateur de VASG (VHDL Analysis and Standardization Group). La société CLSI (CAD Langage Systems Inc.), représentée par *Shahdad* et *Marschner* a préparé une série d'analyses et de recommandations dont a été tirée en Février 1986 la version 7.2 de VHDL, point de départ du futur standard. La collaboration de CLSI au projet était financée par un contrat passé avec l'AFWAL (Air Force Wright Aeronautical Laboratories), représentée par *Hines*. Le standard définitif a été adopté vers le milieu de l'année 1987 [7].

Le langage VHDL-AMS est aussi un standard IEEE (IEEE 1076.1-1999) qui a été développé comme une extension du langage VHDL pour permettre la modélisation et la simulation de circuits et de système analogiques et mixtes analogique numérique. VHDL-AMS constitue un sur ensemble de VHDL, ce qui signifie principalement que:

- toute description VHDL légale l'est aussi en VHDL-AMS et produit les mêmes résultats de simulation ;
- les extensions apportées dans VHDL-AMS conservent les principes VHDL : modularité, déclarations avant usage, typage fort des données, flexibilité, extensibilité. Ces principes concernent à la fois la manière dont le langage est défini et la manière dont un modèle est écrite.

Le langage VHDL-AMS permet de supporter la conception à plusieurs niveaux :

- niveau circuit : modélisation de circuits numériques et analogiques, abstraction possible grâce à des modèles comportementaux de complexités variables (des réseaux de Kirchhoff aux modèles fonctionnels à flot de données) ;
- niveau système : modélisation de systèmes complets (par exemple : une chaîne d'acquisition de données d'un capteur avec traitement numérique) avec prise en compte de l'environnement (par exemple les effets de la température). VHDL-AMS offre en outre un support de base pour la modélisation de systèmes non électriques (capteurs, éléments mécaniques, actionneurs,…).

II- Environnement de travail VHDL-AMS

La figure 2.1 présente l'environnement de travail VHDL-AMS.

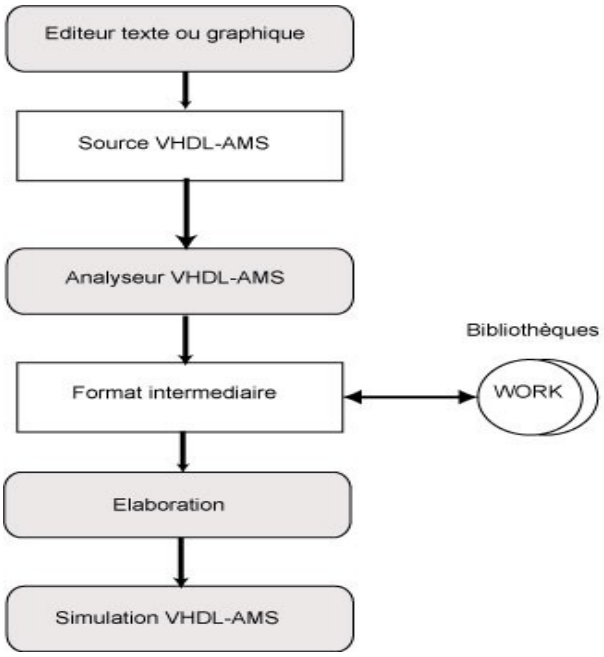


Figure 2.1 : Environnement de travail VHDL-AMS [3]

L'interface graphique peut se réduire à un simple éditeur de texte. La plupart des outils utilise en plus leur éditeur de schémas pour générer automatiquement la squelette d'un modèle VHDL-AMS. Des outils plus avancés permettent de décrire le comportement du système à modéliser sous la forme de machines d'états, de chronogrammes ou de table de vérité.

L'analyseur (ou compilateur) vérifie la syntaxe d'une description VHDL-AMS. Il permet la détection d'erreurs locales, qui ne concernent que l'unité compilée. Plusieurs techniques d'analyses sont actuellement utilisées:

- l'approche compilée produit directement du code machine ou dans certains cas du code C qui sera lui-même compilé. L'objet binaire est alors lié au code objet du simulateur. Cette approche permet de réduire le temps de simulation au détriment du temps d'analyse.
- l'approche interprétée transforme le code source en un pseudo-code qui est interprété par le simulateur. Cette approche réduit le temps d'analyse au détriment du temps de simulation.

Tous les modèles compilés sont placés dans une bibliothèque de travail (*working library*) de nom logique **work** qui est propre à chaque concepteur. Le lien du nom logique avec l'emplacement physique de la bibliothèque dépend de l'outil de simulation utilisé.

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

Le simulateur calcul comment le système modélisé se comporte lorsqu'on lui applique un ensemble de stimulis. L'environnement de test peut également être écrit en VHDL-AMS. Le simulateur permet aussi le débogage d'un modèle au moyen de techniques analogues à celles proposées pour les programmes écrits en Pascal ou C : simulation pas à pas, visualisation de variables, de signaux, modification interactive de valeurs, etc.

III- Structure des modèles VHDL-AMS

Un modèle VHDL-AMS est constitué de deux parties principales :

- la spécification d'entité (**entity**) qui correspond à la vue externe du modèle
- et l'architecture de l'entité (**architecture**) qui est la vue interne du modèle

La structure d'un modèle VHDL-AMS est donnée à la Fig. 2.2.

Au début du code, on fait appel aux bibliothèques (**library**) utiles pour décrire l'architecture en précisant le contenu à exporter. Ces bibliothèques contiennent des fonctions prédéfinies telles que des fonctions arithmétiques, des fonctions mathématiques, des constantes physiques, thermiques ou électromagnétiques, etc. Ces fonctions sont compilés dans des paquetages, et seront déclarés par la commande **use** avant d'être utilisé.

L'entité permet de définir les entrées-sorties du modèle (**port**), à travers lesquels il communique avec son environnement ainsi que les paramètres génériques (**generic**).

L'architecture est constituée d'une zone de déclaration et d'un corps dans lequel on définit le fonctionnement du modèle par l'intermédiaire d'instructions **concurrentes**, **simultanées** ou **séquentielles**. Toutes les instructions peuvent cohabiter offrant ainsi la possibilité d'écrire des modèles pour des circuits analogiques et mixtes avec plusieurs niveaux d'abstraction. Pour une même entité, on peut également écrire plusieurs architectures.

```
    -- bibliothèques utilisées--
    LIBRARY <nom_bibliothèque> ;
    USE <bibliothèque.paquetage1> ;
    USE <bibliothèque.paquetage2>;

    --specification de l'entité--
ENTITY <nom_entité> IS
        GENERIC( <declaration_generic_1>; <declaration_generic_2>;…;
    <declaration_generic_N>
                );
        PORT( <declaration_port_1>; <declaration_port_2>;…;<declaration_port_N>
                );
        [<declarations_variables_globales> ;]
[BEGIN
        <controle_parametres_entree>
]
END ENTITY <nom_entite>;

    --specification de l'architecture--
ARCHITECTURE <nom_arch_1> OF <nom_entite> IS
        <declaration_fonction_procedure>;
        <declaration_constantes>;
        <declaration_terminaux>;
        <declaration_types>;
        <declaration_variables>;

BEGIN
        <type_modele>;
END ARCHITECTURE <nom_archi_1>;

ARCHITECTURE <nom_arch_2> OF <nom_entite> IS …….
ARCHITECTURE <nom_arch_3> OF <nom_entite> IS ….
```

Figure 2.2 : Structure d'un modèle VHDL-AMS

3.1- Déclaration d'entité

On définit dans cette partie l'interface d'un modèle avec le monde extérieur au moyen de ports (**port**). Il existe trois types de ports :

- les ports **signal** : qui définissent des canaux de communication directionnels : entrées (**in**), sorties (**out**) ou bidirectionnels (**inout**) modélisant des signaux logiques.
- les ports **terminal** : qui définissent des points de connexions analogiques adirectionnels pour lesquels pour lesquels les lois de Kirchhoff sont satisfaits. Les terminaux permettent de définir des branches qui elles-mêmes servent de support à la spécification d'équations liant les grandeurs de branches associées, usuellement la tension et le courant pour des systèmes électriques.
- les ports **quantity** : qui définissent des points de connexion analogiques directionnels : entrée (**in**), sorties (**out**) pour lesquels les lois de Kirchhoff ne doivent pas être satisfaits, par exemple pour la modélisation des diagrammes de blocs.

La définition des paramètres génériques peut aussi être faite dans la déclaration d'entité. Ces paramètres serviront à rendre le modèle plus général. La figure 2.3 présente quelques exemples de code pour la déclaration d'entités.

```
--modèle analogique: capacité--
entity capacity is
        generic(cap:real);
        port(terminal n1,n2: electrical);
end entity capacity;

--modèle analogique: multiplieur (sans conservation de
l'energie)--
entity mult is
        port(
        quantity in1,in2: in real;  --opérandes
        quantity reslt: out real;  --résultat

        );
end entity mult;

--modèle numérique: additionneur 1 bit complet--
entity additionneur is
        generic(tprop:time:=0ns); --temps de propagation
        port (
        signal a,b,cin:in bit ;    --entrées :opérandes a et b,
    retenue
        signal s,cout:out bit ;   --sorties: somme,retenue de
    sortie
        );
end entity additionneur ;
```

Figure 2.3 : Code pour la déclaration d'entités

3.2- Déclaration d'architecture

Le langage VHDL-AMS permet de déclarer l'architecture interne du système de deux manières.

- par une description structurelle pour laquelle le modèle est une interconnexion de composants, avec éventuellement un nombre de niveaux hiérarchiques non limités.
- par une description du comportement du circuit dirigé par les évènements, au moyen de types, d'objets et d'instructions appropriés.

3.2.1 La déclaration de l'architecture structurelle

Une architecture structurelle peut être décrite de deux manières : par des déclarations de composants pour définir les besoins de l'architecture. Ces déclarations sont purement locales et ne sont pas nécessairement reliées à des entités de conception particulières, et ensuite par la déclaration de configuration qui est nécessaire pour établir les liens.

3.2.2 La déclaration de l'architecture comportementale

Le comportement d'un circuit est exprimé dans l'architecture grâce à des types, des objets, des instructions simultanées, concurrentes, et séquentielles.

Les types

Un type en VHDL-AMS définit un ensemble de valeurs et les opérations applicables à ces valeurs. Le Tableau 4 présente les différents types et les opérations applicables associées.

Tableau 4 : Les différents types dans VHDL-AMS et les opérations applicables associées [8]

Catégories	Types	Opérations possibles
Scalaires	Numériques : *integer, real*	Logique : not, and, or, nand, nor, xor, xnor Relationnelles: = ,/= , < , <= , > ,>=
	Sous types numériques: *natural, positive*	Arithmétiques : + ,- ,/ ,* ,abs ,** Relationnelles: = ,/= , < , <= , > ,>=
	Enumérés : *Bit, boolean, character*	Logique : not, and, or, nand, nor, xor, xnor Relationnelles: = ,/= , < , <= , > ,>=
	Physiques : *Time, delay_length*	Arithmétiques : + ,- ,/ ,* ,abs ,** Relationnelles: = ,/= , < , <= , > ,>=
Composite	*Array* (tableaux)	Logique : not, and, or, nand, nor, xor, xnor Relationnelles: = ,/= , < , <= , > ,>=
	*Record (*enregistrements)	Relationnelles : = et /=

Les objets

VHDL-AMS possède les principaux objets suivants:

- les constantes (**constant**) qui ont par définition une valeur fixe qui ne peut être modifié.
- les variables (**variable**) qui permettent de stocker une valeur d'un type donné et de modifier cette valeur au moyen d'une instruction d'affectation.
- les signaux (**signal**) qui représentent des formes d'onde logiques sous la forme d'une suite de paires temps/valeur.
- les quantités (**quantity**) qui représentent des fonctions à valeurs réelles du temps, typiquement les inconnus du système d'équations impliqué par le modèle VHDL-AMS. Il y quelques variétés de quantités : les quantités libres, les quantités de branches et les quantités de sources.

  Le langage VHDL-AMS définit aussi des quantités implicites, c'est à dire des quantités qui n'ont pas besoin d'être déclarées, mais qui sont liées à d'autres quantités explicitement déclarées. Par exemple : **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.

Il faut noter que la notation par attribut tick « ‘ » est cumulative, par exemple **Q’dot’dot** représente la dérivée seconde de la quantité Q.

Une liste plus complète de quantités implicites est fournie en annexe.

La figure 2.4 présente un exemple de code pour la déclaration de ces objets.

```
--déclaration de constantes
constant PI : real :=3.1416 ;

--déclaration de variables
variable count : integer ;
variable finished :boolean ;

--déclaration de signaux
signal CLK:bit;

--déclaration de quantités(libres)
quantity q1,q2 : real ;
```

Figure 2.4 : Exemple de code pour la déclaration de constantes, variables, signaux et quantités

Les instructions séquentielles

Un processus (**process**) en VHDL-AMS définit une portion de code dont les instructions sont exécutées en séquence dans l’ordre donné. Les instructions séquentielles possibles dans un processus sont :

- les instructions de contrôle : **if**, **case**, **loop**, **while** ,**for**
- affectation de variables et de signaux (:=,<=)
- la synchronisation : **wait**

Les instructions concurrentes

La base d’un comportement dirigé par les évènements est la notion de processus concurrents. Les **instructions concurrentes** servent à traiter l’information à temps discret. Elles sont évaluées à chaque point de simulation logique en fonction de leur sensibilité à l’évènement courant. Les processus sont des instructions concurrentes, l’affectation des signaux (<=), le **break** ou l’assertion.

Les instructions simultanées

Les instructions simultanées permettent de décrire des équations différentielles algébriques linéaires ou non linéaires. Elles peuvent apparaître partout où une instruction concurrente est légale. Il y a :

- l’instruction simultanée simple : *expression==expression*
- l’instruction simultanée conditionnelle :

if *expression* **use** *expression* ;
elsif *expression* **use** *expression*;
else expression;
end use;

- l'instruction simultanée sélective : **case, use**
- l'instruction procédurale : **procedural**

IV- Les avantages du langage VHDL-AMS

Par rapport aux autres langages de description, VHDL-AMS présente ses principaux atouts pour les points suivants :

- la modélisation comportementale

VHDL-AMS décrit par un ensemble d'équations algébriques et différentielles et par des instructions simultanées le comportement d'un système continu. Les quantités implicites expriment le comportement dynamique des quantités qui leurs sont associés. La description d'un système à temps discrets peut être faite à l'aide des objets **signal**, **quantity** et **terminal** introduits pour décrire la simulation comportementale à temps continu.

- la modélisation mixte

L'instruction **break** permet entre autres d'exprimer la discontinuité dans une simulation à temps continu et couramment utilisée comme moyen de communication ou synchronisation entre la simulation à temps continu et discret..

VHDL-AMS supporte les systèmes conservatifs (loi de Kirchhoff pour les circuits électriques) pour modéliser les systèmes électriques qui sont représentés par les **quantity** et non-conservatifs pour modéliser le flot de données d'un système qui est représenté par les **signal**. Ces deux types forment le système mixte.

- la modélisation mixte et multi-technologie :

VHDL-AMS supporte les systèmes physiques (hydraulique, thermique, etc…) en plus des systèmes électriques. Ces systèmes peuvent être décrits en utilisant les équations algébriques et différentielles. **nature** représente le domaine technologique pour les systèmes conservatifs. Et **quantity across** et **through** préservent la loi de conservation dans le système physique.

- la transparence

VHDL-AMS ne possède pas de modèles primitifs prédéfinis, qui sont déjà implanté. Le concepteur possède la flexibilité de modéliser ses propres systèmes comportementaux ou structurels, et l'utilisateur possède la liberté de modifier les modèles pour les adapter à ses besoins.

V- Exemples de modèles VHDL-AMS

Les instructions du langage VHDL-AMS permettent de présenter le modèle du circuit à plusieurs niveaux d'abstraction. Comme exemples nous allons présenté ici un modèle physique et modèle comportemental.

5.1*Modèle physique : modèle d'une résistance*

La résistance est un composant fondamental dans un système électrique. Pour modéliser la résistance, nous avons besoin de déterminer l'équation qui régit son comportement. La figure 2.4 donne le symbole et l'équation de base qui régit le comportement d'une résistance.

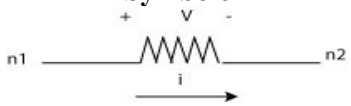
Symbole	**Equation**
	$v = R * i$

Figure 2.5 : Symbole et Equation de la résistance

Ce modèle de base prend juste en considération la loi d'Ohm, R représente la valeur de la résistance. La figure 2.6 présente le code VHDL-AMS de ce modèle de base.

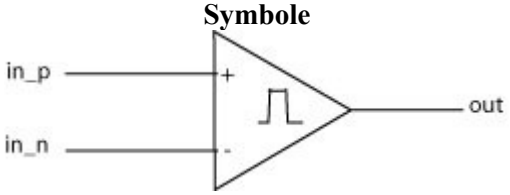
```
-- déclaration de la bibliothèque et des paquetages utilisés--
library IEEE;
use IEEE.electrical_systems.all;

-- declaration de l'entité--
entity resistance is
generic (
        Res : real ); -- valeur de la résistance de type real, pas de valeur par
défaut
port (
        terminal n1, n2 : electrical);
end entity resistance;

--déclaration de l'architecture--
architecture ideal of resistance is
quantity v across i through n1 to n2;

begin
v == Res*i; -- Loi d'Ohm declare comme une instruction simultanée

end architecture ideal;
```

Figure 2.6: Code VHDL-AMS pour le modèle de base de la résistance

5.2*Modèle comportemental d'un circuit : comparateur de tension*

Le comparateur définit ici fournira en sortie un signal numérique de niveau '1' ou '0' selon la comparaison effectuée sur des signaux analogiques en entrée. La figure 2.7 montre le symbole du comparateur et équation qui le régit.

Symbole



Equation

If entrée	Then sortie
in_p>in_n	'1'
In_p<=in_n	'0'

Figure 2.7 : Symbole et équation du comparateur

La figure 2.8 ci-dessous présente le code VHDL-AMS pour le modèle comportemental du comparateur.

```
library IEEE;
use ieee.std_logic_1164.all;
use IEEE.electrical_systems.all;

entity comparateur is
port (
        terminal in_p, in_n : electrical; -- entrées analogiques
        signal output : out std_logic := '1' ); -- sorties logiques
end entity comparateur;

architecture comportementale of comparateur is
        quantity Vin across in_p;
        quantity Vref across in_n;

begin
        process (Vin'above(Vref)) is
                begin
                        if Vin'above(Vref) --
                         then
                        output <= '1' after 1us;
                        else
                        output <= '0' after 1us;
                        end if;
end process;
end architecture comportementale;
```

Figure 2.8: Code VHDL-AMS pour la modèle comportemental du comparateur

VI- La simulation VHDL-AMS

Les différents types d'analyses supportés par VHDL-AMS sont : l'analyse DC, l'analyse temporelle et l'analyse AC petits signaux incluant l'analyse de bruit.

La préparation d'un modèle pour la simulation passe par une phase d'élaboration durant laquelle les valeurs des constantes et des paramètres génériques sont fixées et les modèles sont récupérés dans les bibliothèques de ressources correspondantes. Toutes les instructions concurrentes (équations booléennes) sont prises en charge par le simulateur à évènements discrets (numérique) qui produit des LSP (Logic Simulation Point). Les instructions simultanées (équations différentielles) sont prises en charge par le simulateur

analogique qui produit des ASP (Analog Simulation Point). La simulation du modèle nécessite donc, et dans la plupart des cas, deux noyaux qui doivent être synchronisés.

6.1 Les différentes étapes de la simulation

6.1.1 Phase d'initialisation

Avant le démarrage de l'analyse, il est nécessaire que les objets (variables, signaux, et quantités) du modèle à simuler aient une valeur stable et bien définie. La figure 2.9 présente l'organigramme correspondant à cette phase d'initialisation.

- La première étape consiste à initialiser le temps courant à la valeur 0.0 avec la fonction **now**.
- Tous les objets du modèle prennent ensuite leur valeur initiale qui peut être soit par défaut soit définie dans la déclaration.
- Les processus sont exécutés jusqu'à leur première instruction **wait.**
- Les valeurs des signaux logiques sont alors considérées comme des sources virtuelles et seront considérées avec les sources analogiques pour calculer le point de repos DC de la partie analogique.
- Tant qu'il reste des évènements au temps 0.0, le point de repos DC de la partie analogique est recalculer et les processus sensible aux signaux implicites **Q'above(E)** sera éventuellement réexecuter.
- Les cycles se répètent jusqu'à ce qu'il n'y ait plus d'évènements au temps 0.0.
- La phase d'initialisation se termine alors, et on dit que le système a atteint sont « état quiescent ».

La valeur du signal **domain** représente la phase d'exécution d'un modèle VHDL-AMS : calcul état quiescent (**QUIESCENT_DOMAIN**), analyse temporelle (**TIME_DOMAIN**) ou analyse fréquentielle (**FREQUENCY_DOMAIN**).

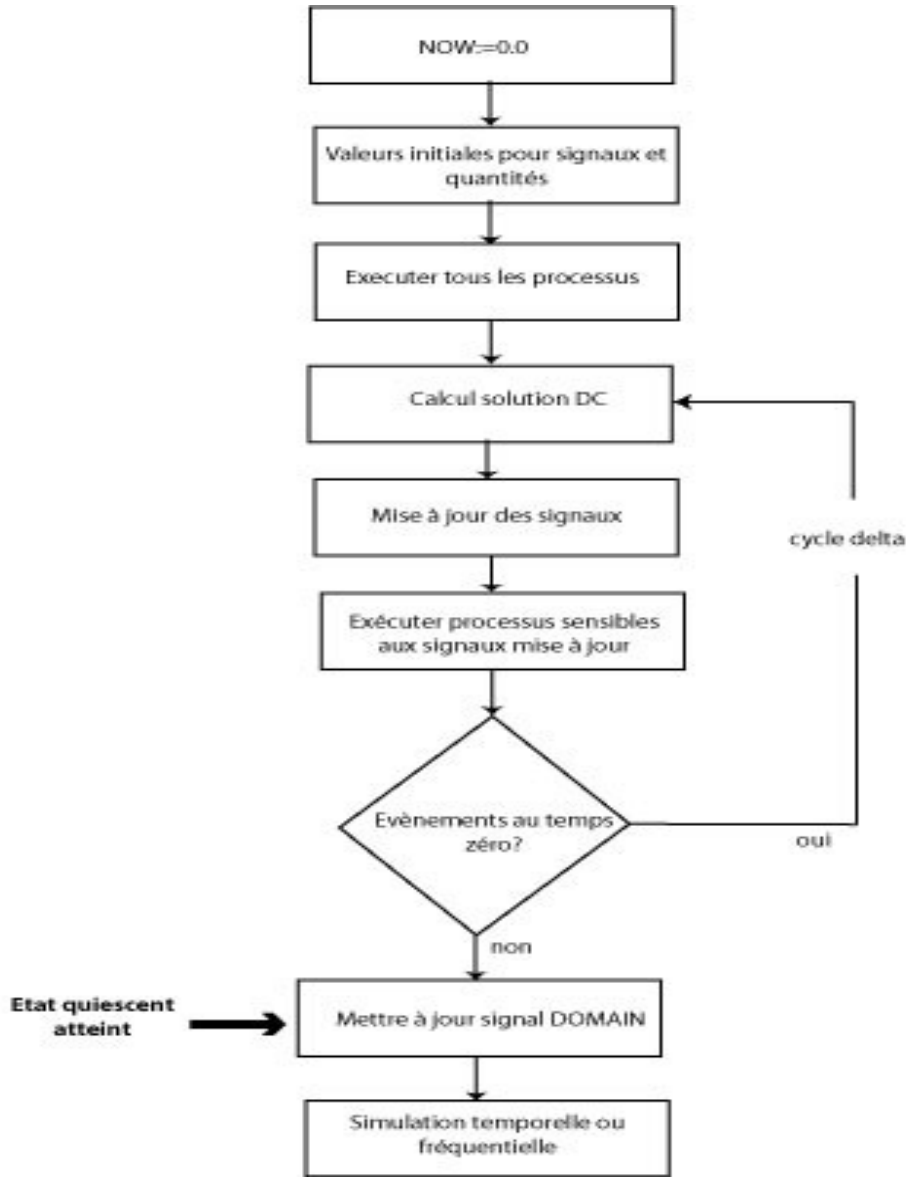


Figure 2.9 : Organigramme de la phase d'initialisation d'une simulation VHDL-AMS

6.1.2 Simulation temporelle

Une simulation temporelle consiste dans la répétition de plusieurs cycles de simulation. Un cycle de simulation est défini entre le temps courant auquel le cycle s'exécute : Tc et le temps du prochain évènement logique. Une fois l'état quiescent atteint, Tc sera égal à 0.0.

- Si le modèle simulé est mixte : tous les signaux au temps 0 sont tous consommés. Le temps Tn est initialement assigné au temps le plus proche pour lequel une ou plusieurs transactions sont prévues.
- Si le modèle simulé est purement analogique, le temps Tn reçoit la valeur **time'high**, ce qui indique qu'il n'y a plus de transaction prévue sur des signaux.

6.1.3 Simulation fréquentielle

L'objectif de l'analyse fréquentielle est d'obtenir les caractéristiques en fréquence d'un circuit lorsqu'il est stimulé par des signaux sinusoïdaux. Ceci revient ainsi à définir des sources d'amplitudes et de phases données et à calculer les amplitudes et les phases résultantes pour les signaux de sortie du circuit.

Le langage VHDL-AMS permet la définition de quantités de sources spectrales selon la syntaxe :

quantity nom {…,…} : type **spectrum** amplitude, phase ;

Le type **spectrum** permet de définir les valeurs réelles de l'amplitude et la phase du signal.

VHDL-AMS définit le calcul des réponses fréquentielles comme :

- le calcul des valeurs fréquentielles (amplitudes et phases) par la résolution du système linéaire.
- le calcul du modèle linéaire.

6.2 *Quelques simulateurs VHDL-AMS*

Le Tableau 5 donne la liste de quelques simulateurs VHDL-AMS existant en version gratuite pour éducation ou démonstration.

Tableau 5 : Quelques simulateurs VHDL-AMS [8]

Compagnie	Nom du simulateur	Adresse
Mentor Graphics	- System Vision - ADVance MS	http://www.mentor.com/products/sm/systemvision/index.cfm
Dolphin Integration	SMASH	http://www.dolphin.fr/medal/smash/smash_overview.html
ANSOFT	Simplorer	http://www.ansoft.com/products/em/simplorer/

Dans ce travail nous avons utilisés le simulateur SYSTEM VISION de Mentor Graphics.

VII- Conclusion

La modélisation des circuits et des systèmes analogiques s'appuie encore bien souvent sur des schémas équivalents à base de primitives SPICE et l'absence de langage de description standardisé a longtemps retardé l'évolution des outils de conception pour l'analogique.

L'approche comportementale de VHDL-AMS offre la souplesse de modélisation qui manque à SPICE. C'est cette caractéristique qui nous a poussé à choisir ce langage comme outils pour la modélisation comportementale établie dans ce travail.

Son atout, allié à la possibilité de simuler des systèmes mixtes devrait rapidement faire de VHDL-AMS la référence dans le domaine.

Chapitre 3: La modélisation comportementale

I- Généralités

1.1 Introduction

La création d'un modèle résulte d'un processus de structuration d'un ensemble de connaissances expérimentales que l'on dispose à propos d'un phénomène ou d'un système physique. Un modèle peut prendre plusieurs formes, mais celles qui nous intéressent sont les modèles mathématiques. Un modèle mathématique définit les fonctions ou plus généralement le comportement d'un système en exprimant des équations définissant des relations entre les variables du système.

Le modèle d'un système est une représentation de son comportement à l'aide de laquelle le simulateur procède à des calculs, un modèle doit être le plus exact possible. La question qui se pose alors est la suivante : comment modéliser les systèmes pour obtenir des modèles fiables répondant à un certain nombre de critères?

Plusieurs méthodes de modélisation ont été établis et offriront aux concepteurs une démarche systématique leur permettant de développer leurs propres modèles en un délai raisonnable.

1.2 *Définition de la modélisation comportementale*

La modélisation comportementale désigne une représentation fonctionnelle de haut niveau, par opposition à une représentation structurelle, et qui est indispensable à la validation de circuits complexes comportant un grand nombre de composants ou sous-systèmes(transistors, diodes, amplificateurs, portes..).

L'objet de la modélisation comportementale est de décomposer le système en un ensemble de blocs fonctionnels, où chaque bloc ou certains d'entre eux sera remplacé par une description fonctionnelle et plus abstraite [4].

1.3 Caractéristiques d'un modèle comportemental

Un circuit électrique communique avec son environnement à travers ses entrées/sorties. Pour modifier ces caractéristiques de transfert, on agit sur les valeurs et les dimensionnements des composants qui le constituent. Un modèle comportemental reprend cette philosophie pour modéliser un circuit électrique. On y définit des entrées/sorties qui sont généralement ceux du circuit modélisé et des paramètres qui permettent de modifier et d'ajuster les caractéristiques de transfert. Un modèle comportemental doit être fiable. Les critères de fiabilité peuvent se résumer en les points suivants :

- une description complète des caractéristiques de transfert du circuit niveau transistor ;
- une bonne précision par rapport au circuit réel ;

- convergence presque sûre des équations dans la modélisation pour les différentes conditions d'opérations et les différents modes de simulations ;
- paramètres génériques permettant d'adapter le modèle pour toute une classe de circuits similaires ;
- un gain de temps en simulation comportementale suffisamment important.

Les langages de modélisation définissent les structures des modèles comportementaux.

II- Méthodologie de modélisation comportementale

On peut distinguer deux notions de modèles : les modèles extraits, développés à partir du schéma transistor et utilisés dans la phase *Bottom-Up* et les modèles génériques développés dans la phase *Top-Down*. A partir de cette distinction, on peut classer les méthodes de modélisation selon deux approches : une approche schématique et une fonctionnelle.

2.1 Approche schématique

L'approche schématique consiste à développer des modèles comportementaux à partir des schémas niveau transistor des circuits à modéliser : soit par l'exploitation du schéma transistor, soit par l'exploitation des résultats de simulation niveau transistor.

2.1.1 Exploitation du schéma transistor

a) Simplification de circuits

La structure fondamentale du circuit est conservée mais certains éléments sont simplifiés ou remplacés par des éléments idéaux. Par exemple, les sources de courant sont remplacées par des sources idéales, certaines structures sont simplifiées, des diodes sont remplacées par des sources de tensions, etc. Pour ce faire on établit un schéma simplifié du circuit et on applique ensuite les modèles simples de transistors et les lois de Kirchhoff pour déterminer la caractéristique de transfert.

b) Simplification du système d'équations différentielles

Une deuxième technique consiste à simplifier le système d'équations différentielles non linéaires obtenu à partir du schéma niveau transistor. Cette simplification est contrôlée par un algorithme d'estimation d'erreur qui permet de réduire le nombre de variables et de paramètres. Cette technique repose sur le développement d'algorithmes de simplification performants et adaptés à tous les circuits analogiques et mixtes.

2.1.2 Exploitation des résultats de simulation

Il s'agit de la détermination de la caractéristique de transfert à partir de l'exploitation des résultats de simulations niveau transistor. Pour cela il faut chercher un modèle mathématique pour les courbes obtenues exprimant les grandeurs de sorties en fonction de celles d'entrées.

2.1.3 Exemple de modélisation schématique

Pour illustrer l'approche schématique nous avons développé un modèle pour l'oscillateur contrôlé en tension (VCO) en utilisant la technique de simplification du schéma transistor. Le schéma niveau transistor du VCO est donné par la Fig. 3.1.

Pour la modélisation on va décomposer le modèles en trois blocs : caractéristiques d'entrée, caractéristiques de transfert et caractéristiques de sortie.

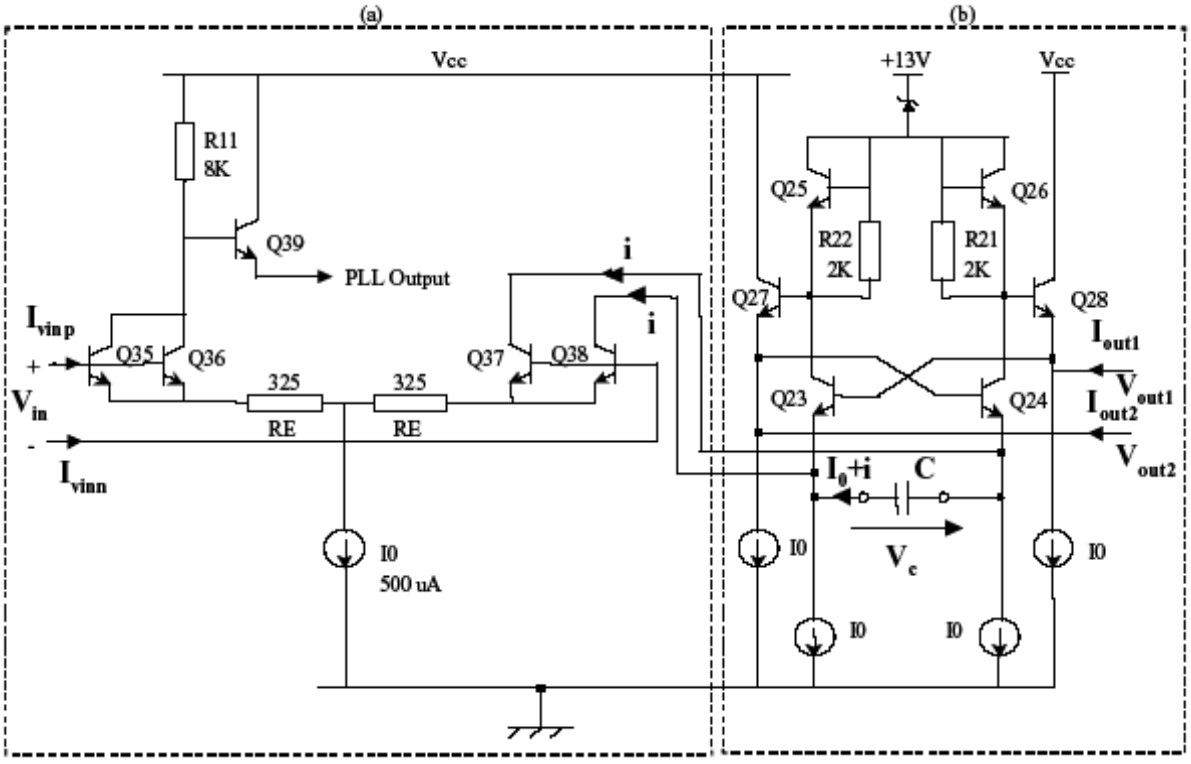


Figure 3.1 : Schéma niveau transistor du VCO [8]

Caractéristiques d'entrée

Les expressions des courants d'entrée I_{vinp} et I_{vinm} sont données par les équations suivantes :

$$I_{vinp} = \frac{Io/\beta}{1+\exp\left(\frac{2V_{in}}{R_{10} * Io}\right)} \qquad (3.1)$$

$$I_{vinm} = \frac{Io/\beta}{1+\exp\left(\frac{-2V_{in}}{R_{10} * Io}\right)} \qquad (3.2)$$

β: Gain en courant des transistors Q35 et Q36.

Caractéristiques de transfert

L'expression du courant i en fonction de la tension d'entrée Vin est donnée par l'équation suivante :

$$i = \frac{I_o}{1 + \exp\left(\frac{2V_{in}}{R_{10} * I_o}\right)} \qquad (3.3)$$

Caractéristiques de sortie

L'oscillateur de relaxation (partie (b)) délivre en sortie une tension avec une fréquence variant linéairement avec le courant i. Son principe de fonctionnement repose sur la charge et décharge de la capacité C. La fréquence du signal de sortie f_{vco} s'exprime en fonction de i et C par l'expression suivante :

$$f_{vco} = \frac{I_o + i}{4CV_{BE}} \qquad (3.4)$$

En se basant sur les équations précédentes, la figure 3.2 montre le modèle schématique du VCO.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;
use IEEE.math_real.all;

entity VCO_schematik is
          generic (ic0,vdd,vm,vp,RE,vbe,beta1,beta2,io,c :real) ;
          port (terminal ip,im,outn,outp:electrical);
end entity VCO_schematik;

architecture comportementale of VCO_schematik is
          quantity vddh, vddl,ic,iinp,iinm,voutn,voutp: real;
          quantity vc across icc through ip to im;
          quantity Vin across iin through ip to im;
          quantity Vout across iout through outp to outn;
begin

  --initialisation
if domain = quiescent_domain
          use icc==ic0 ;
          else vc==icc'integ/c ; --Calcul de la tension aux bornes de la capacité
          end use ;

--Caractéristiques d'entrée suivant Eq 3.1 et 3.2-
iinp==io/beta1/(1.0+ exp((-2.0*(vp-vm))/(RE*io)));
iinm==io/beta2/(1.0+ exp((-2.0*(vm-vp))/(RE*io)));

--Caractéristiques de transfert-
Vin==vp-vm;
ic==io+iinp*beta1/2.0 ; -- calcul du courant i
Vddh==vdd-vbe ;--niveaux haut et bas de la tension de sortie
Vddl==vdd-2.0*vbe ;
```

```
if (icc>=ic)
        use voutn==vddl ;
        voutp==vddh;
                        if (vc>=vbe) -- test de la valeur de la tension aux bornes de la capacité C
                        use icc==-ic ; -- inversion du sens de courant i--
                        else icc==ic ;
                        end use;
else
        voutn==vddh;
        voutp==vddl;
                        if (vc>=vbe) -- test de la valeur de la tension aux bornes de la capacité C
                        use icc==-ic ; -- inversion du sens de courant i--
                        else icc==ic ;
                        end use;
        end use;
vout==voutp-voutn;
end architecture comportementale;
```

Figure 3.2 : Code VHDL-AMS du modèle schématique du VCO

2.2 *Approche fonctionnelle*

Cette approche consiste à analyser la fonction du circuit. Un modèle communique avec son environnement à partir des bornes d'entrées et de sorties. La structure fondamentale du modèle fonctionnel est décomposée en trois parties comme illustré à la Fig. 3.3 :

- la détection des variables d'entrées ;
- le calcul des paramètres des signaux de sorties à partir des variables d'entrées et des paramètres génériques. Ces paramètres génériques sont ajustables de l'extérieur et permettent d'adapter le modèle à une même classe de circuits ;
- et la génération des signaux de sorties.

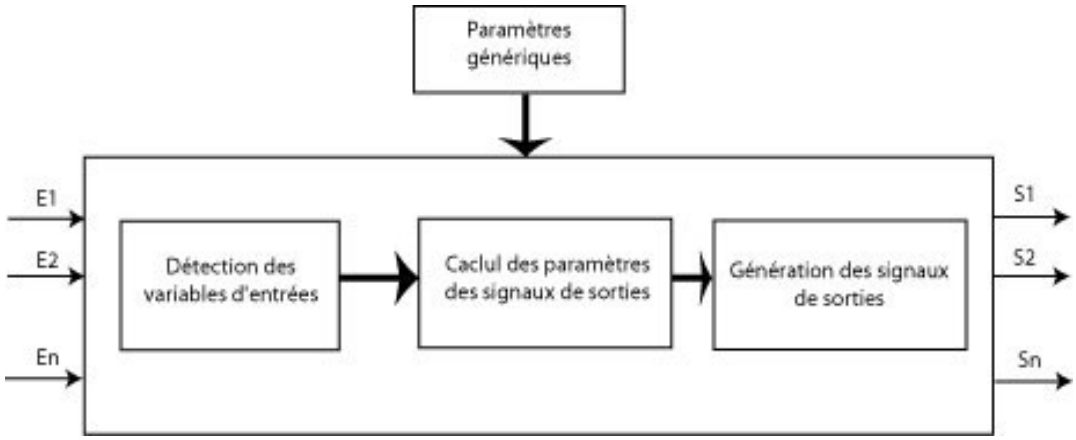


Figure 3.3 : Structure d'un modèle fonctionnel [8]

2.2.1 Détection des variables d'entrées

Cette étape consiste en la détection des informations qu'apportent les signaux d'entrées et qui vont être utiles pour déterminer les paramètres des signaux de sorties. Pour cela il y a cinq types de variables à détecter : la tension d'entrée, le courant d'entrée, la fréquence du signal d'entrée, le front montant et descendant du signal d'entrée et la durée d'une impulsion d'entrée. VHDL-AMS possède les instructions nécessaires pour cette détection.

2.2.2 Calcul des paramètres des signaux de sortie

Il s'agit de déterminer les tensions, les courants, les fréquences ou les formes des signaux de sorties en fonction des paramètres d'entrées et des paramètres génériques. Pour le cas d'un diviseur de fréquence par exemple, la fréquence du signal de sortie est un paramètre à calculer à partir de la fréquence d'entrée, le paramètre générique définit le rapport cyclique.

2.2.3 Génération des signaux de sorties

Pour la génération des signaux de sorties deux cas peuvent se présentés :

- Le signal de sortie dépend directement du signal d'entrée, il s'agit donc d'une génération commandée par le signal d'entrée.

  C'est l'exemple de l'amplificateur : $Vs = A * Ve$, le filtre : $\tau \frac{dVs}{dt} + Vs = A * Ve$

- Le signal de sortie dépend de paramètres caractéristiques du signal d'entrée mais pas directement du signal d'entrée lui-même. Comme pour le cas des générateurs libres sans bornes d'entrées (exemple : générateur de tension sinusoïdale ou carrée), il s'agit de générer le signal en sortie.

2.2.4 Exemple de modélisation fonctionnelle

Nous reprenons ici l'exemple du VCO. Le circuit délivre en sortie une tension périodique dont la fréquence fs varie proportionnellement avec son entrée Vin

$$fs(t) = fo + KoVin(t) \qquad (3.5)$$

Où : fo est la fréquence centrale du VCO et Ko son gain

La figure 3.4 donne le code VHDL-AMS du modèle fonctionnel du VCO. La fréquence centrale fo du VCO et son gain Ko sont considérés comme paramètres génériques.

```
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

entity VCO is
  generic (
    Kv         : real;   -- Gain du VCO [Rad/s/Volt]
    Fc         : real;    -- Fréquence centrale [Hz]
    Vc         : voltage ;    -- Amplitude de la tension d'entrée [Volts]
    Vcmin      : voltage;    -- Amplitude minimum [Volts]
    Vcmax      : voltage;    -- Amplitude maximum [Volts]
    Vout_ampl  : voltage;    -- Amplitude de la tension de sortie [Volts]
    Vout_offset : voltage     -- offset de la tension de sortie [Volts]
    );
  port (
    terminal v_inp,  v_outp,: electrical);
end entity VCO;

-------------------------------------------------------------------------------
-- VCO Equation:
-- Fout = Fc + Kv*Vin
-------------------------------------------------------------------------------
architecture comportementale of VCO is
  quantity vout across iout through v_outp to ELECTRICAL_REF;
  quantity vctrl across v_inp to ELECTRICAL_REF;
  quantity phi : real;
  quantity vtmp : real;

begin

--limites de la zone linéaire de la caractéristique du VCO
  if vctrl > Vcmax use
    vtmp == Vcmax;
  elsif vctrl < Vcmin use
    vtmp == Vcmin;
  else
    vtmp == vctrl;
  end use;

  if domain = quiescent_domain use
    phi     == 0.0;
  else

  -- Calcul de la fréquence de sortie en Rad/s
    phi'dot == Fc + Kv*(vtmp-Vc);
   end use;

-- Génération de la tension de sortie
vout == Vout_offset + Vout_ampl*cos(math_2_pi*phi);

end architecture comportementale;
```

Figure 3.4 : Code VHDL-AMS du modèle fonctionnel du VCO

2.3 Simulation des modèles

Les deux types de modèles du VCO élaborés précédemment ont été simulés pour comparer les performances de deux approches de modélisation comportementale.

Une tension d'entrée sinusoïdale d'amplitude 12 V est appliquée à l'entrée. Les caractéristiques du VCO ont été fixés comme suit :

Kv= 100. 10^3 rad/s.V
Fc= 100 kHz

Nous avons effectué une analyse temporelle d'une durée de 1ms. Les résultats des simulations des deux modèles sont présentés aux Fig. 3.5 et 3.6.

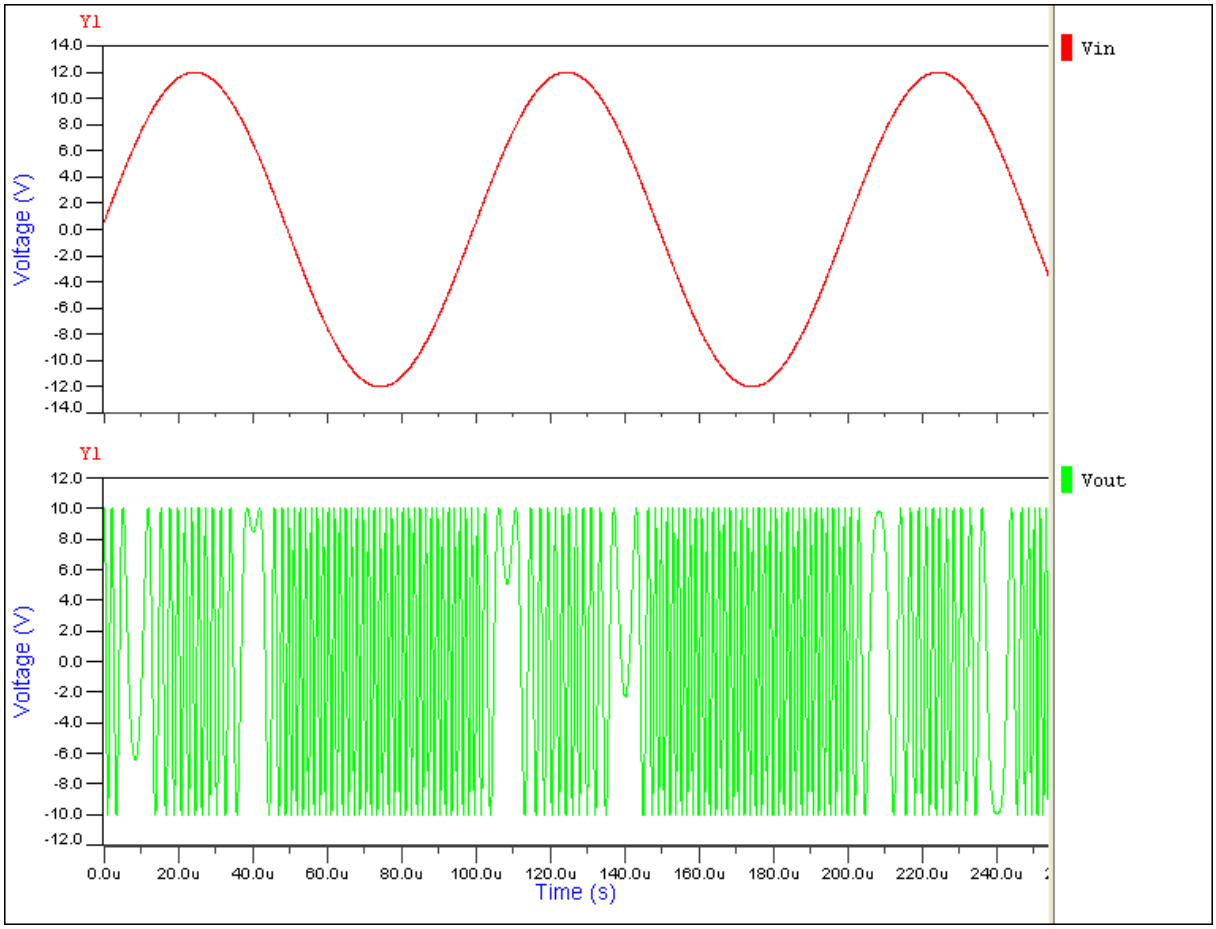


Figure 3.5 : Réponse temporelle du modèle schématique du VCO

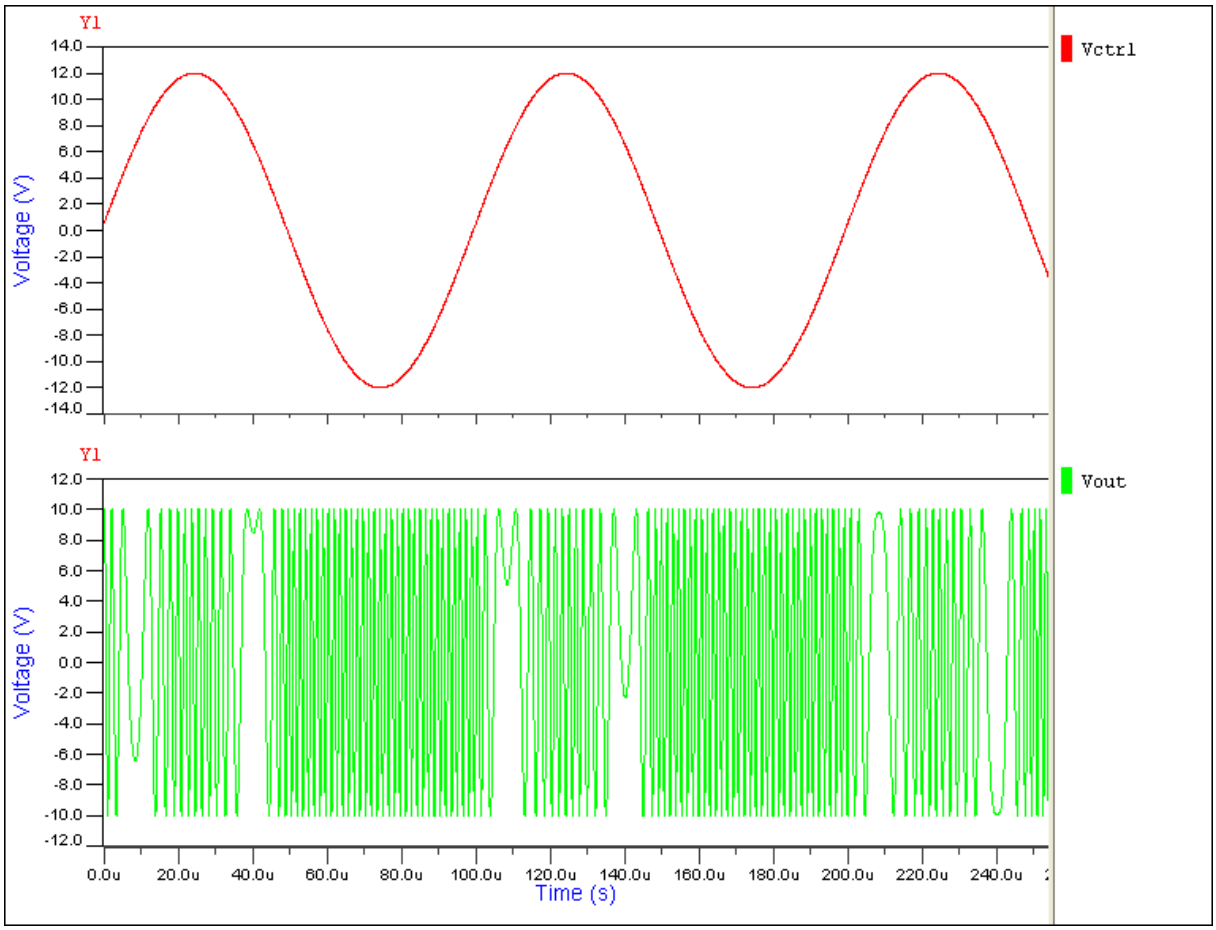


Figure 3.6 : Réponse temporelle du modèle fonctionnel du VCO

2.4 Comparaison entre les approches schématique et fonctionnelle

Les réponses des deux modèles sont presque identiques. Néanmoins pour une analyse temporelle de 1ms la durée de la simulation pour l'approche schématique est moins rapide que celui du modèle fonctionnel. Le Tableau 6 présente la comparaison faite sur les temps de simulation des deux types de modèles.

Tableau 6: Comparaison des temps de simulation

Type de modèle	Temps CPU pour la simulation
Modèle schématique	40 s 530 ms
Modèle fonctionnel	7 s 561 ms

Le constat est immédiat, la simulation d'un modèle développé selon une approche schématique est moins rapide par rapport au modèle fonctionnel. Ceci pour deux raisons : il utilise les modèles simples des transistors en appliquant les lois de Kirchhoff ce qui aboutit à des équations différentielles complexes et d'autre part sa taille augmente avec la taille du circuit. Un modèle fonctionnel présente une meilleure performance sur la rapidité parce qu'il repose sur l'analyse de la fonction du circuit et non sur l'application des modèles simples des transistors.

D'autre part, ils n'existent pas aussi des paramètres génériques compromettant la réutilisation du modèle schématique et la détermination des paramètres pour la simulation repose sur la résolution des différentes équations régissant les caractéristiques d'entrée, de transfert et de sortie. Le modèle établi avec l'approche fonctionnel peut être réutilisé du fait qu'il utilise des paramètres génériques.

III- Conclusion

La modélisation comportementales permettent de :

- résoudre les problèmes de convergences des simulateurs qui sont conditionnées par le nombre de composants dans le système.
- concevoir des circuits de plus grande qualité et répondant aux spécifications demandées.

Deux méthodes de modélisation comportementale ont étés présentées dans ce chapitre : l'approche schématique et fonctionnelle. Après la comparaison faite entre ces deux approches nous avons adopté l'approche fonctionnelle pour développer le modèle comportemental du synthétiseur de fréquence que nous allons présenter dans le chapitre suivant.

Chapitre 4: Application à la modélisation comportementale d'un synthétiseur de fréquence

I- Généralités

1.1 Introduction

Les synthétiseurs de fréquence permettent de synthétiser une bande de fréquences à partir d'une fréquence de référence appliquée à l'entrée.

Un synthétiseur de fréquence est caractérisé par :

- sa plage de fréquence ;
- son pas de synthèse qui est égale à la largeur du canal de l'application considérée ;
- son temps d'établissement qui correspond au temps que mets le synthétiseur pour passer d'un état stable à un autre ;
- et ses bruits de phase qui proviennent de l'oscillateur.

La synthèse de fréquences est utilisée à des fins diverses : générer une horloge synchronisant des processus numériques de traitement du signal comme les conversions analogique numérique et numérique analogique, l'échantillonnage des signaux,…. dans le domaine analogique comme oscillateur local pour translater le signal d'une fréquence à l'autre à une autre.

Les champs d'application de la synthèse de fréquence nécessitent de la part du synthétiseur des qualités différentes : précision, stabilité, vitesse d'acquisition consommation, coût de fabrication….Aujourd'hui, les systèmes basés sur la boucle à verrouillage de phase (PLL) sont les plus populaires pour réaliser une telle fonction car ils possèdent la plupart des qualités citées précédemment et sont devenus des architectures maîtrisées. [5]

Pour ce travail nous nous sommes intéressés aux synthétiseurs de fréquence fractionnaires à base de boucle à verrouillage de phase.

1.2 Principe de fonctionnement :

La figure 4.1 présente l'architecture de base d'un synthétiseur de fréquence utilisant une boucle à verrouillage de phase (PLL).

La PLL est un système bouclé dans lequel la phase d'un signal d'entrée est asservie à la phase d'un signal de référence.

Le boucle constituant le synthétiseur de fréquence est composée des éléments suivants :

- un comparateur phase fréquence suivie d'une pompe de charge ;
- un filtre passe-bas ;

- un oscillateur contrôlé en tension ;
- un diviseur de fréquence.

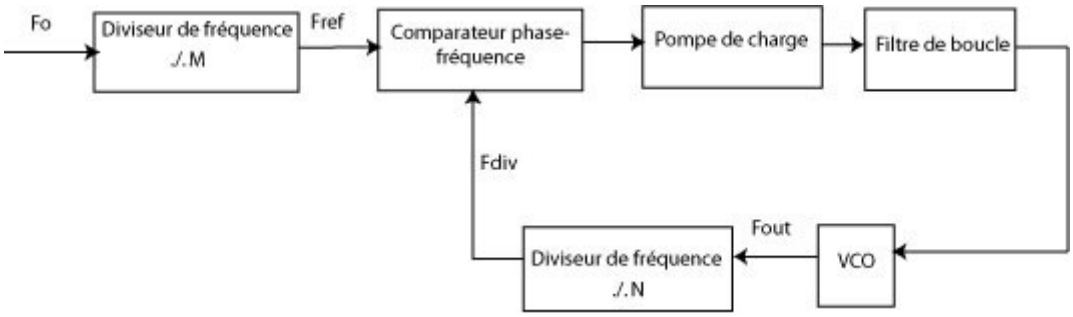


Figure 4.1 : Architecture de base d'un synthétiseur de fréquence utilisant un PLL [8]

- La sélection du canal à synthétiser se fait en agissant sur le rapport de division N.
- Pour un déphasage entre les tensions V(Fdiv) et V(Fref), le comparateur de phase génère une tension d'erreur.
- Cette tension est filtrée par le filtre de boucle et sa valeur moyenne pilote le VCO.
- Lorsque la boucle est verrouillée la tension de sortie du filtre est constante et Fdiv égale à Fref.
- Pour réaliser la synthèse de la fréquence dans le cas où on utilise un diviseur fractionnaire (à la place du diviseur par N), on fait varier le rapport de division N en utilisant un compteur programmable dans la boucle de retour.
- Un diviseur de fréquence fractionnaire est souvent constitué d'un accumulateur et d'un diviseur N/N+1.

II- Modélisation des blocs constituant le synthétiseur de fréquence

Nous avons développé le modèle comportemental de chaque bloc. Pour les modèles analogiques, les entrées et sorties sont définies comme étant des terminaux électriques. Pour les modèles numériques, les interfaces sont définies comme des signaux de type bit.

2.1 *Comparateur de phase*

Le comparateur de phase doit donner en sortie une information sur le déphasage entre le signal de sortie du VCO et le signal d'entrée de la boucle.

Un comparateur de phase seul ne permet pas d'asservir correctement la PLL. Il est nécessaire d'utiliser un comparateur phase fréquence car le comparateur de phase ne peut asservir correctement que des signaux déphasés mais de mêmes fréquences.

Le comparateur de phase est linéarisé autour du point de fonctionnement de la boucle défini par fo, ce qui veut dire qu'il sera caractérisé par un coefficient souvent noté Kd défini par :

$$K_d = \frac{U_{moyen}}{\Phi}[V/rad] \qquad (4.1)$$

Où : U_{moyen} est la valeur moyenne de la tension de sortie
Φ : le déphasage entre les signaux d'entrée

Le modèle comportemental qui sera élaboré ici est celui d'un comparateur phase fréquence numérique. La figure 4.2 présente le schéma d'un comparateur de phase fréquence numérique.

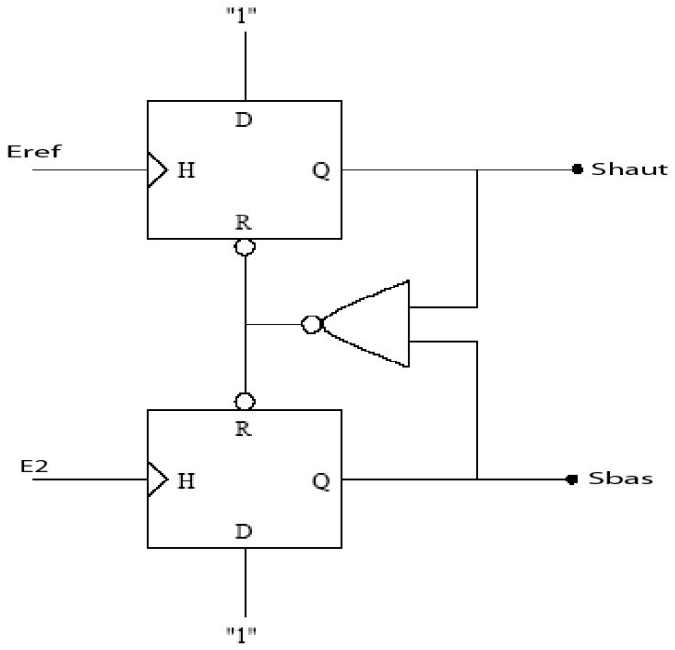


Figure 4.2 : Comparateur de phase fréquence à porte NAND

2.1.1 Principe de fonctionnement

On utilise souvent des comparateurs OU exclusif ou à porte NAND, le principe reste le même. Le chronogramme du comparateur phase fréquence est présenté à la fig. 4.3

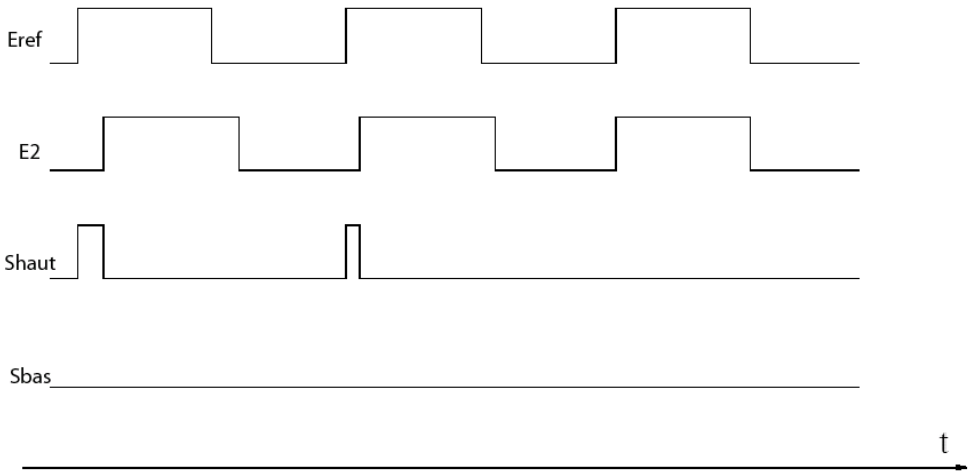


Figure 4.3 : Chronogramme du comparateur phase fréquence

Le comparateur génère en sortie deux signaux : *Shaut* et *Sbas*.

- Supposons à l'origine des temps que tous les signaux sont nuls.
- A l'avènement du front montant du signal de référence *Eref* : *Shaut* est au niveau haut et *Sbas* au niveau bas. *Eref* est en avance de phase sur l'entrée E2.
- *Shaut* reste à 1 jusqu'au front montant de E2.

- Lorsque *Eref* est en retard de phase par rapport à E2, *Shaut* est au niveau bas. A cet instant *Sbas* passe au niveau haut.
- Lorsque les deux signaux d'entrée sont synchronisés, *Shaut* et *Sbas* sont tous les deux au niveau bas.
- La détection de l'avance ou du retard de phase d'un signal par rapport à un autre se fait sur les fronts montants.
- La sortie *Shaut* nous donne le déphasage des deux signaux de d'entrée.

2.1.2 Modèle comportemental du comparateur de phase

Suivant le principe de fonctionnement expliqué précédemment, nous avons élaboré le modèle du comparateur. Le code VHDL-AMS du modèle élaboré est donné à la Fig. 4.4.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity comp_phase_frequence is
  generic (VH,VB,TM,TD: real);
  port (terminal Eref,E2,Sh,Sb: ELECTRICAL);
end entity comp_phase_frequence;

architecture comportementale of comp_phase_frequence is

  constant Vmoyenne: real := (VH+VB)/ 2.0;
  signal etat: real := 0.0;
  signal controle_1: bit := '0';
  signal controle_2: bit := '0';

  quantity V1 across I1 through Eref to ELECTRICAL_REF;
  quantity V2 across I2 through E2 to ELECTRICAL_REF;
  quantity Vsh across Ish through Sh to ELECTRICAL_REF;
  quantity Vsb across Isb through Sb to ELECTRICAL_REF;

begin

I1==0.0;
I2==0.0;
```

```
phase_frequence : process
  begin
    -- détection des fronts montants

  wait until (V1'above(vmoyenne)'event)  or (V2'above(vmoyenne)'event);

  -- Eref et E2 synchronisé

  if (V1'above(vmoyenne)=true) and    (controle_1='0')
     and (V2'above(vmoyenne)=true) and    (controle_2='0')
  then
     (controle_1) <='1';
     (controle_2) <='1';
     etat <= 0.0;
  else

  -- Eref en avance de phase sur E2

  if (V1'above(vmoyenne)=true) and (controle_1='0') then controle_1<='1';
  if ((etat=0.0) or (etat=1.0)) then etat <=1.0;
  else etat<=0.0;
  end if;
  end if;

  -- Eref en retard de phase sur E2

  if (V2'above(vmoyenne)=true) and (controle_2='0') then controle_2<='1';
  if ((etat=0.0) or (etat=-1.0)) then etat <=-1.0;
  else etat<=0.0;
  end if;
  end if;

  end if;

  -- Détection des fronts descendants

  if (V1'above(vmoyenne)=false) then controle_1<='0';
  end if;

  if (V2'above(vmoyenne)=false) then controle_2<='0';
  end if;

 end process phase_frequence;

-- Generation des signaux de sorties

if etat>0.0 use Vsh==VH*etat'ramp(tm,td);
else Vsh==VB;
end use;

 if etat<0.0 use Vsb==VH*etat'ramp(tm,td);
 else Vsb==VB;
 end use;
end architecture comportementale;
```

Figure 4.4 : Code VHDL-AMS du modèle comportemental du comparateur de phase

Le processus « phase_fréquence » au début de ce modèle servira à détecter à l'aide de la variable « *etat* » le déphasage entre les deux signaux d'entrée *Eref* et *E2*. L'activation du processus se fait à chaque détection du front montant de *Eref* (*contrôle_1=1*) ou E2(*contrôle_2=1*).

Tableau 7 : Valeur du variable « *etat* » en fonction du déphasage

Déphasage	**Valeur de *« etat »***
Eref et E2 synchronisés	0
Eref en avance sur E2	Incrémenté de 1, puis =1 si était à 0 ou 1 =0 si était à -1
Eref en retard sur E2	Décrémenté de 1, puis =-1 si était à 0 ou -1 =0 si était à 1

La génération du signal de sortie dépendra de la valeur de « *etat* ».

Tableau 8 : Valeur de la sortie en fonction de la valeur du variable « *etat* »

etat	**Sortie**
positif	Sh
négatif	Sb

2.1.3 Simulation du modèle du comparateur phase fréquence

Un exemple de simulation du modèle avec SystemVision est illustré par la Fig. 4.5 où sont présentées les deux entrées Eref et E2 et la sortie Sh.

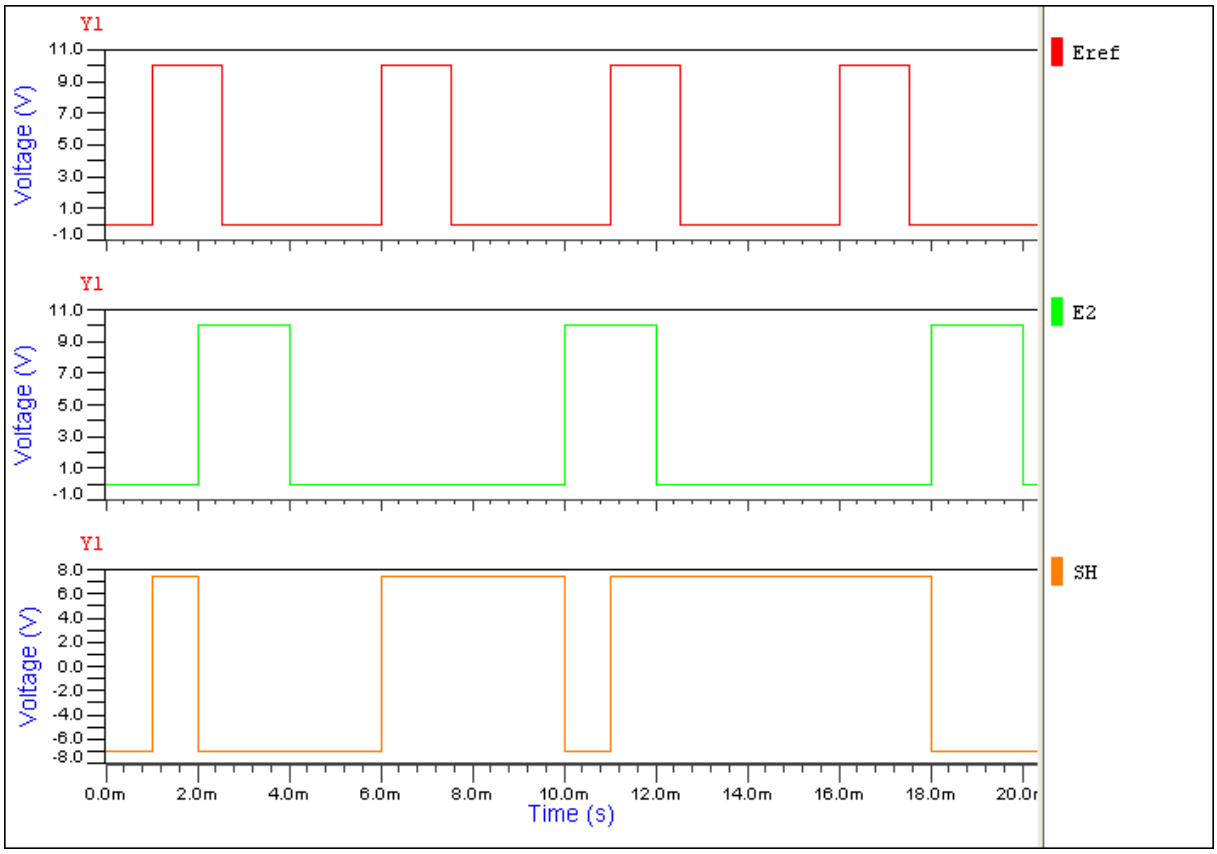


Figure 4.5 : Simulation du modèle du comparateur de phase

2.2 *Pompe de charge*

On utilise souvent aujourd'hui des comparateurs phase fréquence avec sortie en courant appelés aussi comparateurs à pompe de charge. Le résultat de la comparaison en tension est converti en courant par la pompe de charge avant d'être intégré dans le filtre.

Le courant moyen en sortie du comparateur à pompe de charge est sensiblement proportionnel au déphasage entre les deux signaux d'entrée. Le comparateur à pompe de charge sera alors caractérisé par sa transmittance :

$$K_d = \frac{I_{moyen}}{\Phi}[A/rad] \qquad (4.2)$$

Où I_{moyen} la valeur moyenne du courant de sortie
Φ : le déphasage entre les signaux d'entrée

2.2.1 Principe de fonctionnement

Généralement, la pompe de charge est constituée de deux sources de courant contrôler par les deux sorties du comparateur phase fréquence selon le circuit de la Fig. 4.6.

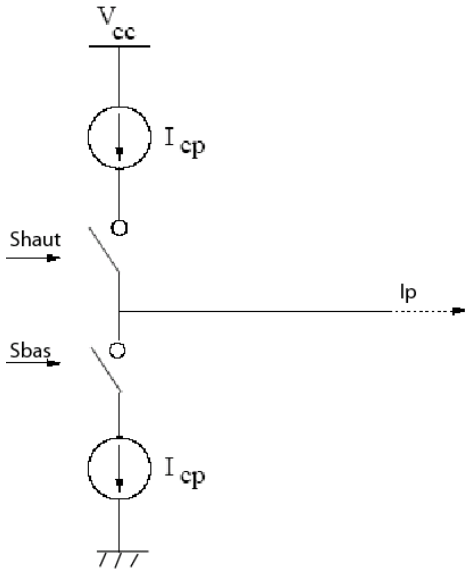


Figure 4.6 : Schéma de la pompe de charge [6]

- Lorsque la sortie *Shaut* du comparateur est à son niveau haut, la pompe de charge génère un courant positif.
- Lorsque la sortie *Sbas* du comparateur est à son niveau haut, la pompe de charge génère un courant négatif.

2.2.2 Modèle comportemental de la pompe de charge

Le processus « pompe__charge » au début de ce modèle permet la détection de l'état des deux entrées et génère en fonction les deux signaux de contrôle : *controle_1* et *controle_2* selon le Tableau 9.

Tableau 9 : Valeur des variables de contrôle en fonction de l'état de la tension d'entrée

V1 (resp. V2)	***controle_1* (resp. *controle_2*)**
Front montant	1
Front descendant	0

Le courant de sortie dépend des valeurs des variables *controle_1* et *controle_2*.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity pompe_charge is
        generic (VH,VB,TM,TD,ampli_courant: real);
        port (terminal Eref,E2,S:ELECTRICAL);
end entity pompe_charge;

architecture comportementale of pompe_charge is

 constant Vmoyenne: real := (VH+VB)/2.0;
 signal controle_1: real := 0.0;
 signal controle_2: real := 0.0;
 signal diff_controle: real := 0.0;
 quantity V1 across Ih through Eref to ELECTRICAL_REF;
 quantity V2 across Ib through E2 to ELECTRICAL_REF;
 quantity Vs across Iout through S to ELECTRICAL_REF;

begin
 pompe_charge : process

  begin

                                        -- détection des variables d'entrée

        wait until (V1'above(Vmoyenne)'event) or (V2'above(Vmoyenne)'event);
  -- détection front montant V1
        if (V1'above(Vmoyenne)=true) and (controle_1=0.0) then controle_1<=1.0;
        end if;
  -- détection front descendant de V1
        if (V1'above(Vmoyenne)=false)  then controle_1<=0.0;
        end if;
 -- détection front montant V2
        if (V2'above(Vmoyenne)=true) and (controle_2=0.0) then controle_1<=1.0;
        end if;
  -- détection front descendant de V2
        if (V2'above(Vmoyenne)=false)  then controle_2<=0.0;
        end if;

 end process pompe_charge;

 Ih==0.0;
 Ib==0.0;
 diff_controle<=controle_1-controle_2;

                                        -- Génération de Ip courant de sortie
 Iout/ampli_courant==diff_controle'ramp(TM,TD);

end architecture comportementale;
```

Figure 4.7 : Code VHDL-AMS du modèle comportemental de la pompe de charge

2.2.3 Simulation du modèle de la pompe de charge

Les sorties du comparateur de phase fréquence ont été repris pour la simulation du modèle de la pompe de charge. L'amplitude du courant de sortie a été fixé à 5 V. Les résultats obtenus sont présentés à la Fig. 4.8.

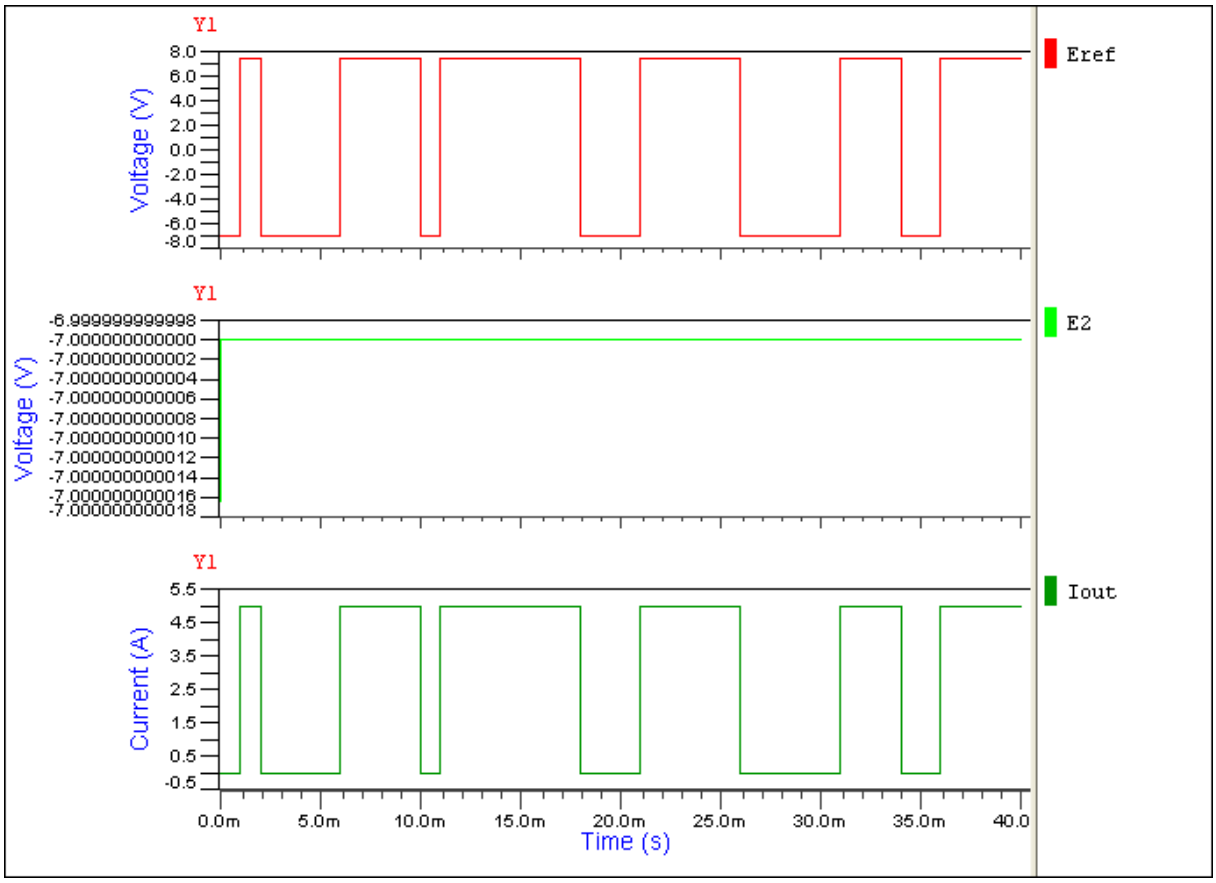


Figure 4.8 : Simulation du modèle de la pompe de charge

2.3 *Filtre passe-bas*

Le filtre de boucle utilisé est un filtre passe-bas d'ordre 2. L'ordre du filtre introduit une intégration supplémentaire aux basses fréquences et augmente ainsi la précision sans dégrader la marge de phase et la stabilité.

Les différents paramètres du filtre sont choisis de la façon suivante :

- le gain dépend du gain du comparateur de phase
- la fréquence de coupure devra être au moins une décade en dessous de la fréquence centrale de la boucle.

2.3.1 Principe de fonctionnement

Le comportement du filtre passe-bas du second ordre peut être défini par sa fonction de transfert. Cette fonction de transfert du filtre présenté dans la Fig 4.9 s'exprime par :

$$H(p) = H_0 \cdot \frac{\omega_p^2}{p^2 + \left(\frac{\omega_p}{Q_p}\right)p + \omega_p^2} \qquad (4.3)$$

Où Ho : gain du filtre en DC
ω_p : sa pulsation propre
Q_p : son facteur de qualité

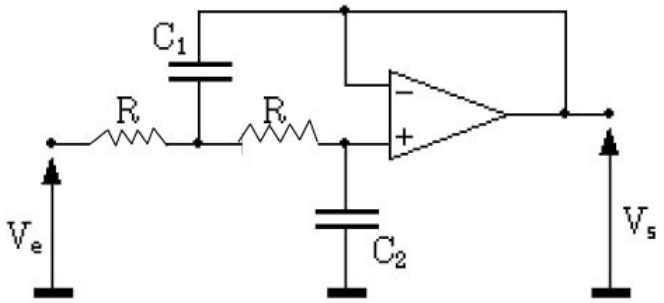


Figure 4.9 : Exemple d'un filtre passe bas actif d'ordre 2

2.3.2 Modèle comportemental du filtre

```
library IEEE;
use IEEE.electrical_systems.all;
use IEEE.math_real.all;

entity filtre_passebas is
  generic (
    Fp   : real := 1.0e6;   -- fréquence propre [Hz]
    Ho    : real := 1.0;     -- gain du filtre
    Q  : real := 1.0);      -- facteur qualité
  port (terminal input  : electrical;
        terminal output : electrical);
end entity filtre_passebass;

architecture comportementale of filtre_passebas is

-- détermination des paramètres de la fonction de transfert
  quantity vin across input to electrical_ref;
  quantity vout across iout through output to electrical_ref;
  constant wp  : real := math_2_pi*Fp;  -- calcul de la pulsation propre en Rad
  constant num : real_vector := (wp*wp, 0.0);       -- calcul du numérateur
  constant den : real_vector := (wp*wp, wp/Q, 1.0); -- calcul du dénominateur

begin

--Génération du signal de sortie
  vout == Ho * vin'ltf(num, den);
end architecture comportementale ;
```

Figure 4.10 : Code VHDL-AMS du modèle comportemental du filtre passe bas

2.3.3 Simulation du modèle du filtre

Nous avons simulé le modèle du filtre pour Fp= 120 kHz, K=1, Q=1 ; la Fig.4.11 montre le diagramme de Bode du filtre.

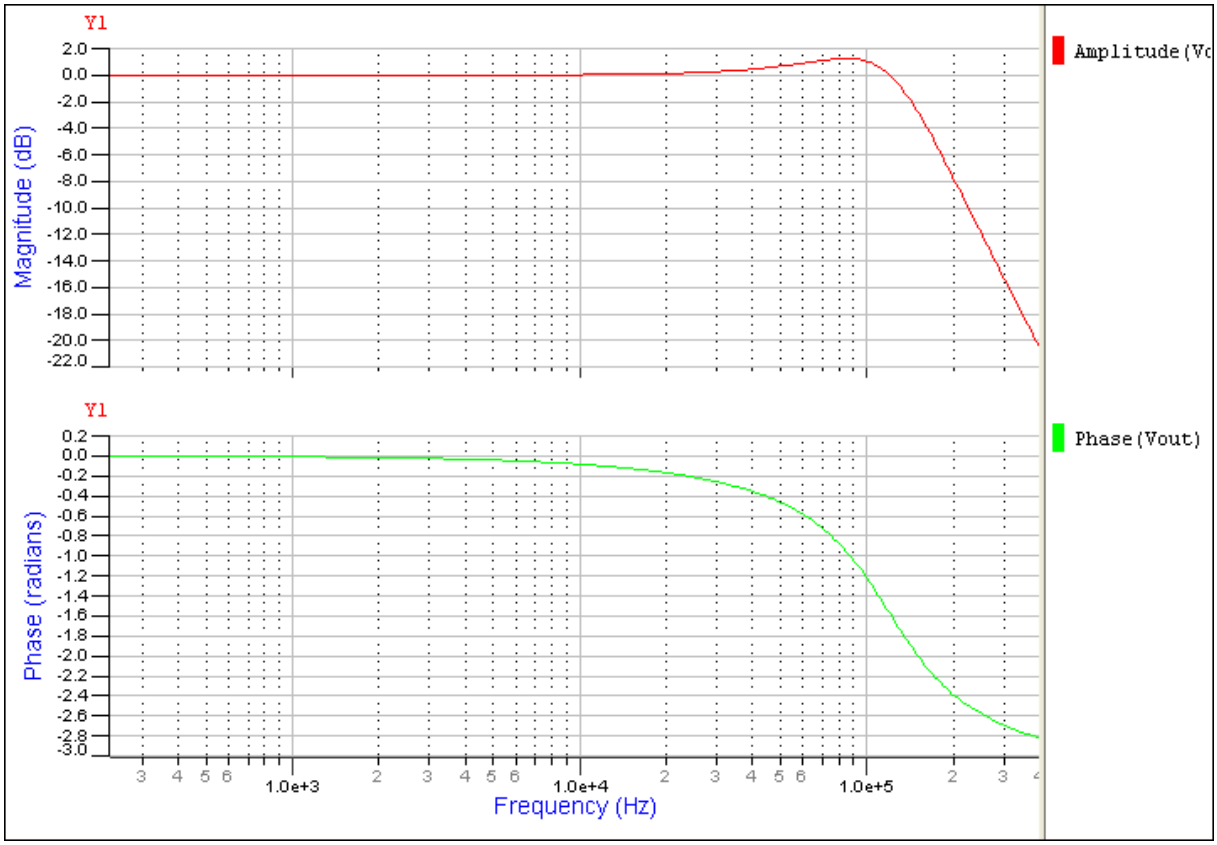


Figure 4.11 : Simulation du modèle du filtre passe-bas

2.4 Diviseur de fréquence fractionnaire

Un diviseur de fréquence fractionnaire permet de diviser la fréquence d'un signal d'entrée par un entier ou par un réel. La division fractionnaire est réalisée en faisant passer le rapport de division de N à M de façon dynamique de sorte que le rapport de division moyen après T périodes corresponde à un réel non entier. L'utilisation d'un diviseur de fréquence fractionnaire dans la synthèse de fréquence permet de générer toute une plage de fréquence.

Le diviseur de fréquence fractionnaire développé dans ce travail est constitué d'un accumulateur et d'un diviseur de fréquence N/M. L'accumulateur sert à déterminer le rapport de division de N à M.

2.4.1 Principe de fonctionnement

La figure 4.12 illustre le principe de la division fractionnaire.

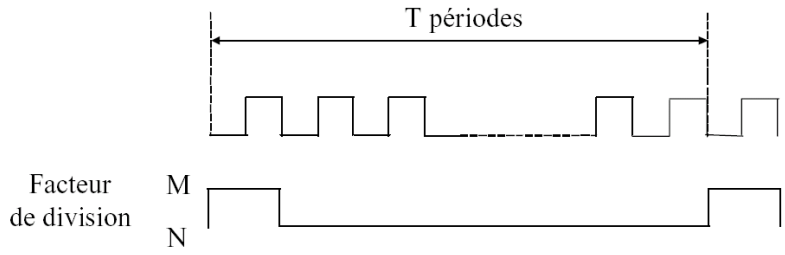


Figure 4.12 : Chronogramme du diviseur de fréquence fractionnaire [8]

La fréquence du signal d'entrée est divisée une fois par *M* toutes les *T* périodes et par *N* pendant *T-1* périodes. Si on divise *K* fois par *M*, on divise *T-K* fois par *N*. *K* étant le paramètre qui permet de fixer le rapport de division fractionnaire et est compris entre 0 et T.

Le facteur de division fractionnaire est alors comme suit :

$$D_{frac} = M\left(\frac{K}{T}\right) + N\left(\frac{T - K}{T}\right) \qquad (4.4)$$

2.4.2 Modèle comportemental du diviseur fractionnaire

Comme expliqué précédemment, nous allons développé les modèles de l'accumulateur et du diviseur N/M pour modéliser le diviseur fractionnaire.

a) Accumulateur

L'accumulateur va générer le signal permettant de commander le diviseur N/M et d'effectuer le choix du rapport de division.

Le circuit possède trois entrées :

- entrée horloge
- entrée T : qui fixe la période du signal de sortie tel que :

$$T_s = T * T_k$$

Ts étant la période du signal de sortie et *Tk* la période de l'horloge.

- et l'entrée K : qui est le nombre du front montant par période de sortie.

La figure 4.13 présente le schéma bloc de l'accumulateur.

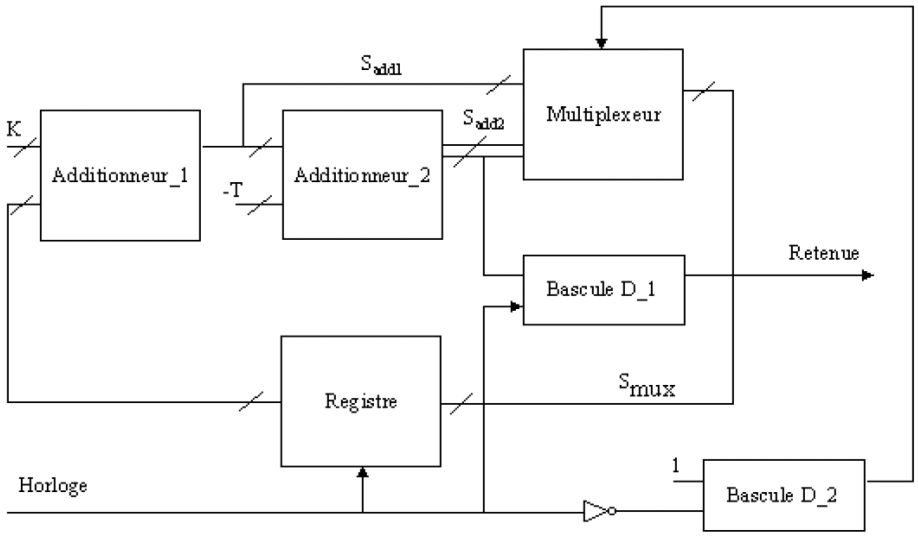


Figure 4.13 : Schéma bloc de l'accumulateur [8]

- L'*additionneur_1* additionne la valeur de l'entrée K avec la sortie du registre initialement à 0.
- L'*additionneur_2* additionne la sortie de l'*addtionneur_1* avec –T.
- Les sorties de deux additionneur *Sadd1* et *Sadd2* sont multiplexées : si *Sadd2 >0* elle sera récupérée à la sortie du multiplexeur, sinon on récupère *Sadd1*
- L'une des deux sorties sera alors stockée dans le registre pour être additionnée de nouveau avec la valeur de K et le cycle recommence.
- Le dernier bit de *Sadd2* est dirigé vers la bascule *D_1* et sera le signal de commande du diviseur N/M.

Le code VHDL-AMS du modèle comportemental de l'accumulateur est donné à la Fig. 4.14.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity accumulateur is
        generic (VH,VB,T,K: real); -- T et K paramètres génériques
        port (terminal Tin:ELECTRICAL; signal retenue: out bit);
end entity accumulateur;

architecture comportementale of accumulateur is

 constant Vmoyenne: real := (VH+VB)/2.0;
 signal compteur_1: real := 0.0;
 signal mux: real := 0.0;
 quantity compteur_2: real := _20.0;
 quantity Vin across Tin to ELECTRICAL_REF;

begin
 accumulateur : process

  begin
     -- premier additionneur
    compteur_1<=K+ mux ;
```

```
  -- multiplexage et détermination de la sortie « retenue »
        wait on Vin'above(Vmoyenne) ;

        if compteur_2>=0.0 then mux<=compteur_2 ;
        retenue<='1' ;
        else mux<=compteur_1 ;
        retenue<='0' ;
        end if ;

 end process accumulateur;

  -- deuxième additionneur
        compteur_2==compteur_1-T ;

end architecture comportementale;
```

Figure 4.14 : Code VHDL-AMS du modèle comportemental de l'accumulateur

Le processus « accumulateur » est activé à chaque front montant du signal d'horloge. La sortie *retenue* est un signal numérique qui va commander le diviseur N/M.

La figure 4.15 montre le résultat de la simulation du modèle du comparateur pour une période de l'horloge de 1 ms , T égal à 20 et K égal à 4.

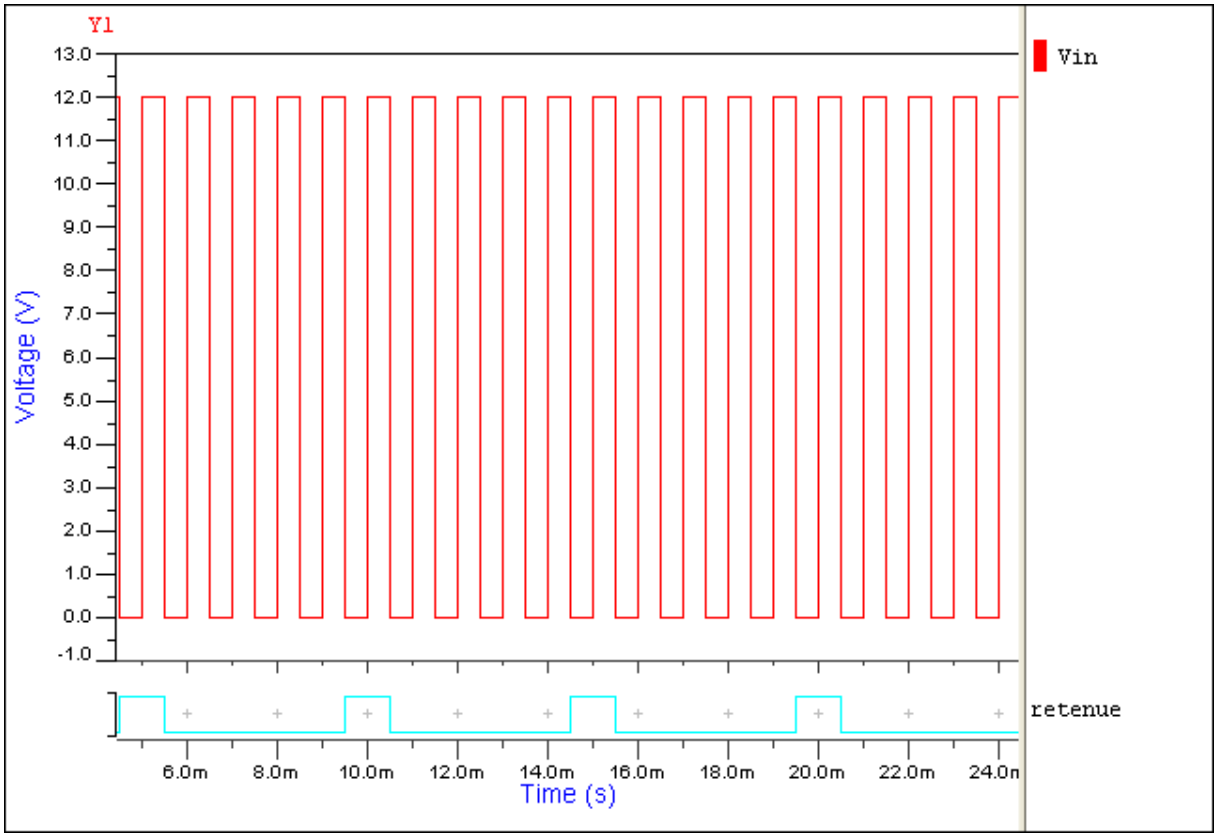


Figure 4.15 : Simulation du modèle de l'accumulateur

b) Diviseur N/M

La figure 4.16 montre le chronogramme d'un diviseur N/N+1.

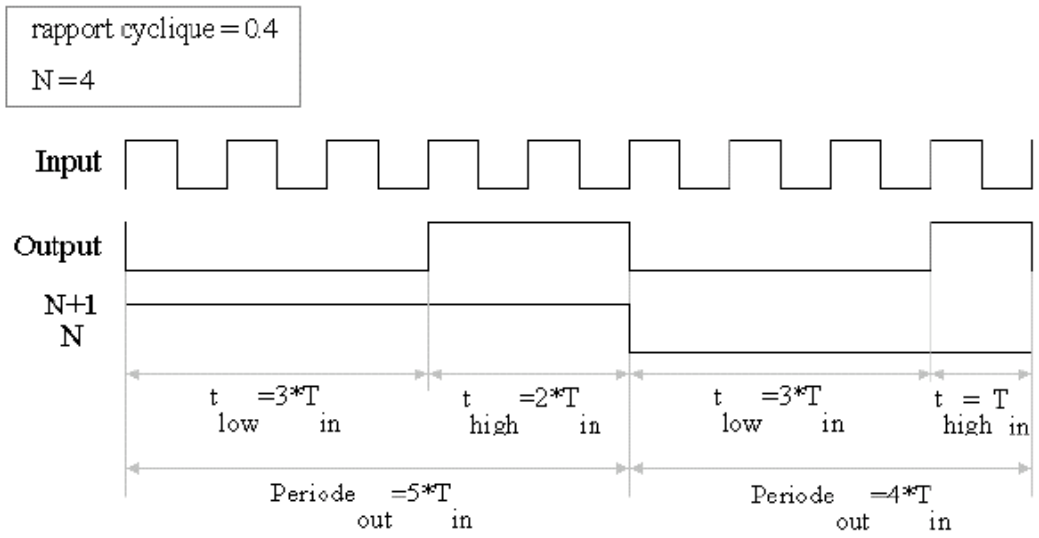


Figure 4.16 : Chronogramme d'un diviseur de fréquence N/N+1 [8]

Lorsque l'entrée de commande (sortie de l'accumulateur) est au niveau haut, le rapport de division est égal à N+1. Lorsqu'elle est au niveau bas le rapport de division est égal à N.

Le code VHDL-AMS du modèle comportemental de l'accumulateur est donné à la Fig. 4.17. Le principe consiste à détecter les fronts montants du signal d'entrée pour incrémenter un compteur et déterminer, suivant la valeur du rapport de division et du rapport cyclique, l'état du signal de sortie.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity diviseur is
        generic (VH,VB,TM,TD,N,M,rapport_cyclique: real);
        port (terminal Tin1,Tout:ELECTRICAL; signal retenue: in bit);
end entity diviseur;

architecture comportementale of diviseur is

  constant Vmoyenne: real := (VH+VB)/2.0;
  signal compteur: real := -1.0;
  signal Vs1 :real = VB;
  quantity rapport_division :real ;
  quantity Vin across Iin through Tin to ELECTRICAL_REF;
  quantity Vout across Iout through Tout to ELECTRICAL_REF;
```

```
begin

iin==0.0 ;

 diviseur : process

  begin

    -- détection du front montant du isgnal d'entrée
        wait on Vin'above(Vmoyenne) ;
  -- incrémentation du compteur
        compteur<=compteur + 1.0 ;

  -- détermination du rapport de division
        if retenue='1' use rapport_division==M ;
        else rapport_division==N ;
        end use;

  -- calcul d'un signal intermédiaire qui définira l'état de la sortie
        if (compteur<2.0*(rapport_division)*rapport_cyclique) then Vs1<=VB
        else Vs1<=VH
        end if ;

   - remise à zero du compteur
        if (compteur>2.0*(rapport_division-1.0) then compteur<=0.0 ;
        end if ;

 end process diviseur;

  -- génération du signal de sortie
Vout==Vs1'ramp(TM,TD) ;

end architecture comportementale;
```

Figure 4.17: Code VHDL-AMS du modèle comportemental du diviseur N/M

c) **Diviseur fractionnaire**

Pour établir le modèle du diviseur fractionnaire, on va associer les modèles de l'accumulateur et du diviseur N/M définis précédemment de manière structurel. Le code VHDL-AMS du modèle comportemental de l'accumulateur est donné à la Fig. 4.18.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity diviseur_fractionnaire is
        generic (VH,VB,TM,TD,N,M,rapport_cyclique,T,K: real ;);
        port (terminal Tin1,Tin2,Tout:ELECTRICAL);
end entity diviseur_fractionnaire;

architecture structurel of diviseur_fractionnaire is

 signal retenue: bit:='0';

begin

    -- appel au modèle de l'accumulateur
        accumulateur : entity work.accumulateur(comportementale)
                generic map (T=>T,K=>K,VH=>VH,VB>=VB)
                port map (Tin=>Tin1, retenue=>retenue) ;

    -- appel au modèle du diviseur N/M
        diviseur : entity work.diviseur(comportementale)
                generic map (TM=>TM,TD=>TD,VH=>VH,VB>=VB,N=>N,M=>M,rapport_cyclique=>
                rapport_cyclique)
                port map (Tin=>Tin2,Tout=>Tout, retenue=>retenue) ;

end architecture structurel;
```

Figure 4.18: Code VHDL-AMS du modèle comportemental du diviseur fractionnaire

Pour la simulation du modèle du diviseur fractionnaire, nous avons repris les paramètres génériques de l'accumulateur vu ci-dessus : T est égal à 20 et K est égal à 4, pour une période du signal d'horloge de 1ms. Le diviseur utilisé est un diviseur 4/5. Pour une période de 0,2 ms de l'entrée Vin le résultat de la simulation du diviseur fractionnaire est donné à la Fig.4.19.

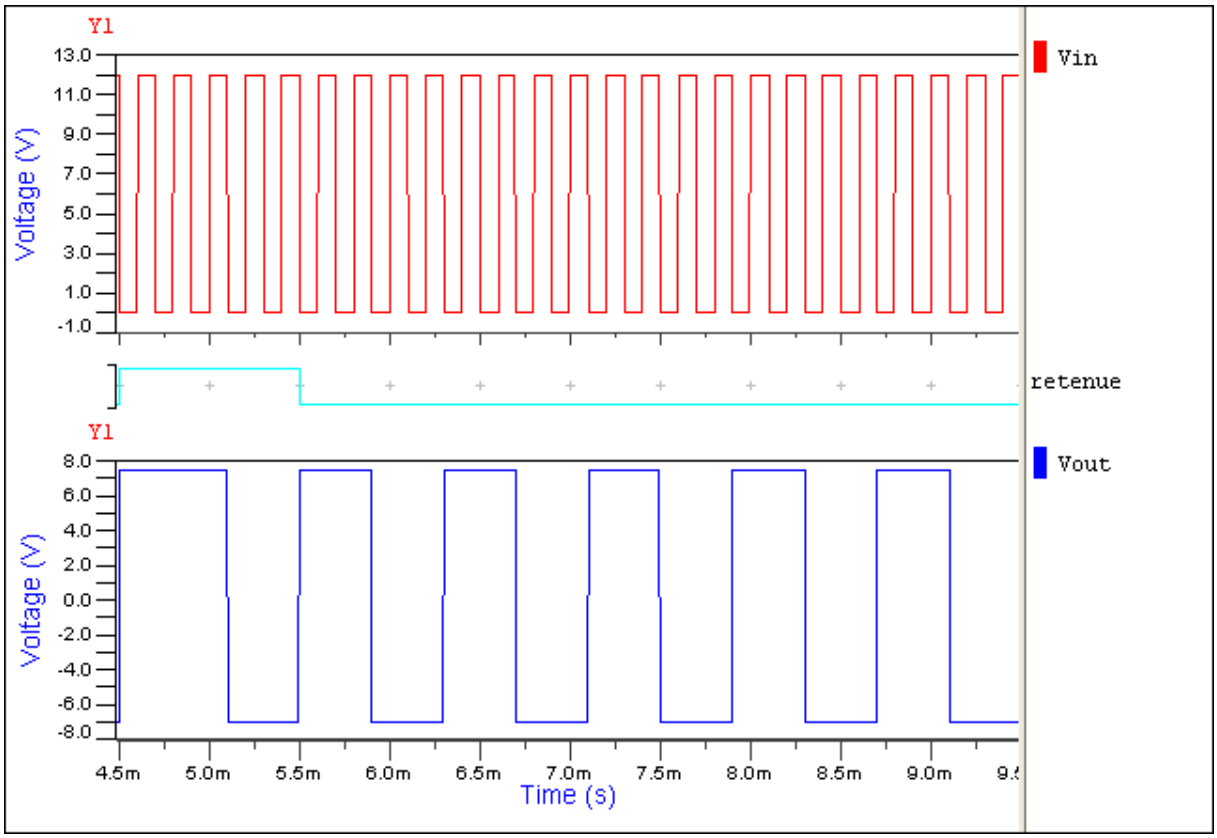


Figure 4.19 : Simulation du modèle comportemental du diviseur fractionnaire

2.5 *Oscillateur contrôlé en tension (VCO)*

2.5.1 Principe de fonctionnement

En l'absence de signal d'entrée, le VCO oscille à sa fréquence propre f_0. En appliquant une tension d'entrée *Vin*, la fréquence de sortie varie proportionnellement à cette tension suivant l'équation suivante :

$$f_{vco} = f_0 + K_{vco} v_{in} \qquad (4.5)$$

Kvco est le gain du VCO exprimé en rad/s.V

2.5.2 Modèle comportemental du VCO

Nous allons reprendre le modèle fonctionnel du VCO déjà élaboré au Chapitre 3 § 2.2.4.

III- Simulation du modèle comportemental du synthétiseur de fréquence

Les modèles développés dans le paragraphe précédent permettront d'élaborer le modèle du synthétiseur de fréquence fractionnaire de manière structurel. La figure 4.20 présente le code VHDL-AMS du modèle comportemental du synthétiseur en faisant appel aux modèles de comparateur phase fréquence, pompe de charge, filtre passe bas, diviseur fractionnaire et VCO.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity synthetiseur_frequence is
        generic (
        M1: real;       --Facteur de division du diviseur à l'entrée^
        rapport_cyclique1 : real ;
        VH: real;       --  Paramètres du comparateur de phase
        VB: real;
        TM: real;
        TD: real;
        ampli_courant: real ;  -- Amplitude du courant à la sortie de la pompe de charge
        Fp : real := 1.0e6;   -- fréquence propre du filtre de boucle [Hz]
        Ho: real := 1.0;      -- gain du filtre
        Q : real := 1.0);      -- facteur qualité
        N: real;       -- Facteur de division du diviseur fractionnaire
        M: real;       -- Facteur de division du diviseur fractionnaire
        rapport_cyclique2: real;
        T: real;
        K: real;
        Kv        : real;   -- Gain du VCO [rad/s.Vt]
        Fc        : real;    -- Fréquence centrale [Hz]
        Vc        : voltage ;    -- Amplitude de la tension d'entrée [Volts]
        Vcmin     : voltage;    -- Amplitude minimum [Volts]
        Vcmax     : voltage;    -- Amplitude maximum [Volts]
        Vout_ampl  : voltage;    -- Amplitude de la tension de sortie [Volts]
        Vout_offset : voltage     -- offset de la tension de sortie [Volts]

        );
        port ( terminal Tin,Tvco : electrical);

end entity synthetiseur_frequence;

architecture structurel of synthetiseur_frequence is
        terminal Tref: electrical;
        terminal Tdiviseur: electrical;
        terminal Thaut: electrical;
        terminal Tbas: electrical;
        terminal Tpompe: electrical;
        terminal Tfiltre: electrical;
```

```
begin


Diviseur_M: entity work.diviseur(comportementale)
                generic map (VH=>VH,VB=>VB,TM=>TM,TD=>TD,
                diviseur=>M1,rapprot_cyclique=>rapport_cyclique1)
                port map (Tin=>Tin, Tout=>Tref) ;

Diviseur_fractionnaire : entity work.diviseur_fractionnaire ( comportementale)
                generic map (T=>T,N=>N,M=>M,rapprot_cyclique=>rapport_cyclique2,K=>K,
                VH=>VH,VB=>VB,TM=>TM,TD=>TD)
                port map (Tin1=>Tref,Tin2=>Tvco,Tou=>Tdiviseur) ;

Comparateur : entity work.comp_phase_frequence (comportementale)
                generic map (VH=>VH,VB=>VB,TM=>TM,TD=>TD)
                port map (Eref=>Tref,E2=>Tdiviseur,Sh=>Thaut,Sb=>Tbas);

Pompe:          entity work.pompe_charge (comportementale)
                generic map (VH=>VH,VB=>VB,TM=>TM,TD=>TD,
                ampli_courant=>ampli_courant)
                port map (Eref=>Thaut,E2=>Tbas,S=>Tpompe);


Filtre:         entity work.filtre_passebas (comportementale)
                generic map (Fp=>Fp,Ho=>Ho,Q=>Q)
                port map (input=>Tpompe,output=>Tfiltre


VCO :           entity work.VCO (comportementale)
                generic map (Kv=>Kv,FC=>FC,Vc=>Vc,Vcmin=>Vcmin,Vcmax=>Vcmax,
                Vout_ampl=>Vout_ampl,Vout_offset=>Vout_offset)
                port map (v_inp=>Tfiltre,v_outp=>Tvco) ;

end architecture structurel;
```

Figure 4.20 : Code VHDL-AMS du modèle comportemental du synthétiseur de fréquence

3.1 Détermination des paramètres génériques du synthétiseur de fréquence

Le schéma équivalent du synthétiseur fractionnaire modélisé est présenté à la Fig. 4.21.

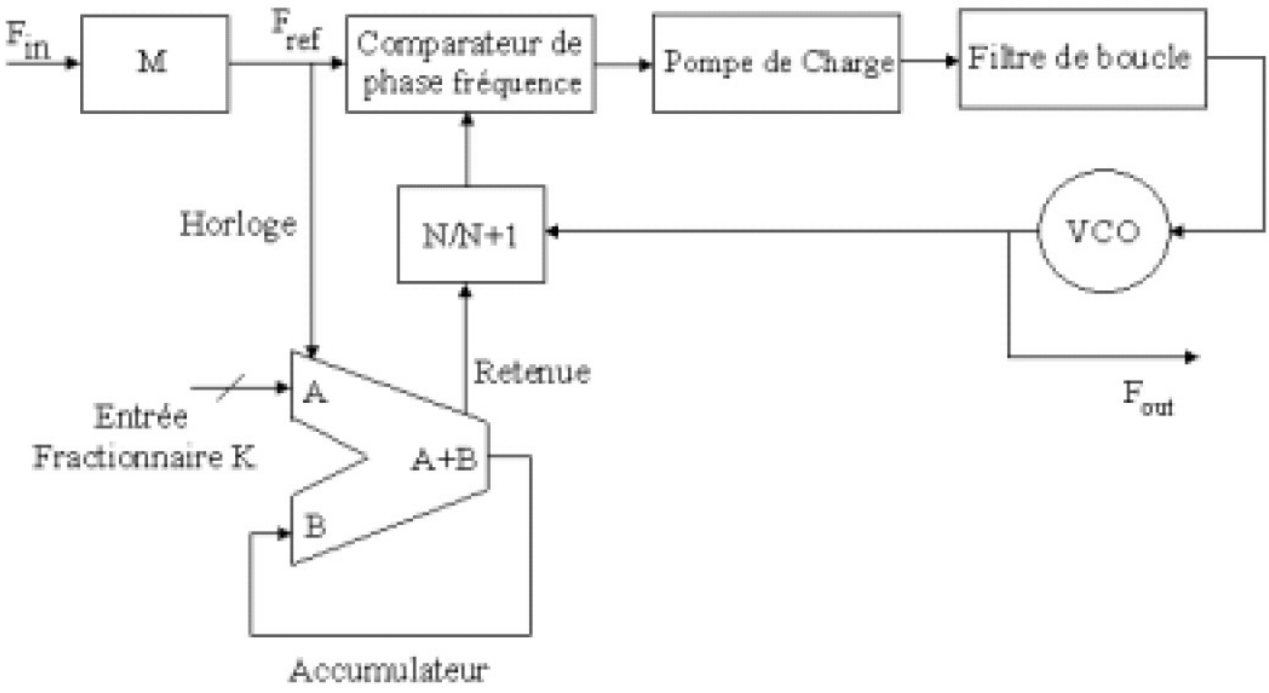


Figure 4.21 : Schéma du synthétiseur fractionnaire

Pour l'application nous avons déterminé les paramètres génériques pour une utilisation du synthétiseur répondant à la norme UMTS (Universal Mobile Telecommunication System). Les caractéristiques de la norme UMTS se résument comme suit :

Bande d'émission/réception : 1920 MHz – 1980 MHz
Nombre de canaux : 12
Bande allouée par canal : 5 MHz
Sensibilité du récepteur : -100 dBm
Sensibilité de l'émetteur : 0.25 W

Le tableau 10 résume les différents paramètres génériques du synthétiseur fractionnaire. La fréquence à la sortie du VCO varie de 320 MHZ à 330 MHz correspondant à la bande d'émission/réception divisée par 6. Les calculs correspondants à la détermination des différents paramètres sont fournis en annexe.

Tableau 10 : Valeurs des paramètres génériques du synthétiseur de fréquence

Blocs	**Paramètres**	**Valeurs**
Diviseur par M	- Facteur de division : **M1** - Rapport Cyclique : **rappot_cyclique1**	6 0.2
Diviseur fractionnaire	- Fréquence de division : **T** - Entrée fractionnaire : **K** - Rapport de division : **N/M (où M=N+1)** - Rapport cyclique : **rapport_cyclique2**	20 10 19/20 0.2
Pompe de charge	- Amplitude courant : **ampli_courant**	100 µA
Filtre passe bas	- Fréquence propre du filtre : **Fp** - Gain : **Ho** - Facteur qualité : **Q**	117 kHz 1 1
VCO	- Gain : **Kv** - Fréquence propre : **Fo**	50 MHz 325 MHz

3.2 *Simulation du synthétiseur de fréquence*

3.2.1 Environnement de simulation de SYSTEM VISION

L'environnement de simulation de SYSTEM VISION de Mentor Graphics est présenté à la Fig. 4.22. Dans cet environnement, on peut visualiser le code VHDL-AMS du circuit, sa structure du projet, les listes des processus, des objets et des paramètres du circuit, les fichiers de sortie et les détails de la simulation. Une fenêtre graphique permet de visualiser les courbes de simulation selon la Fig. 4.23.

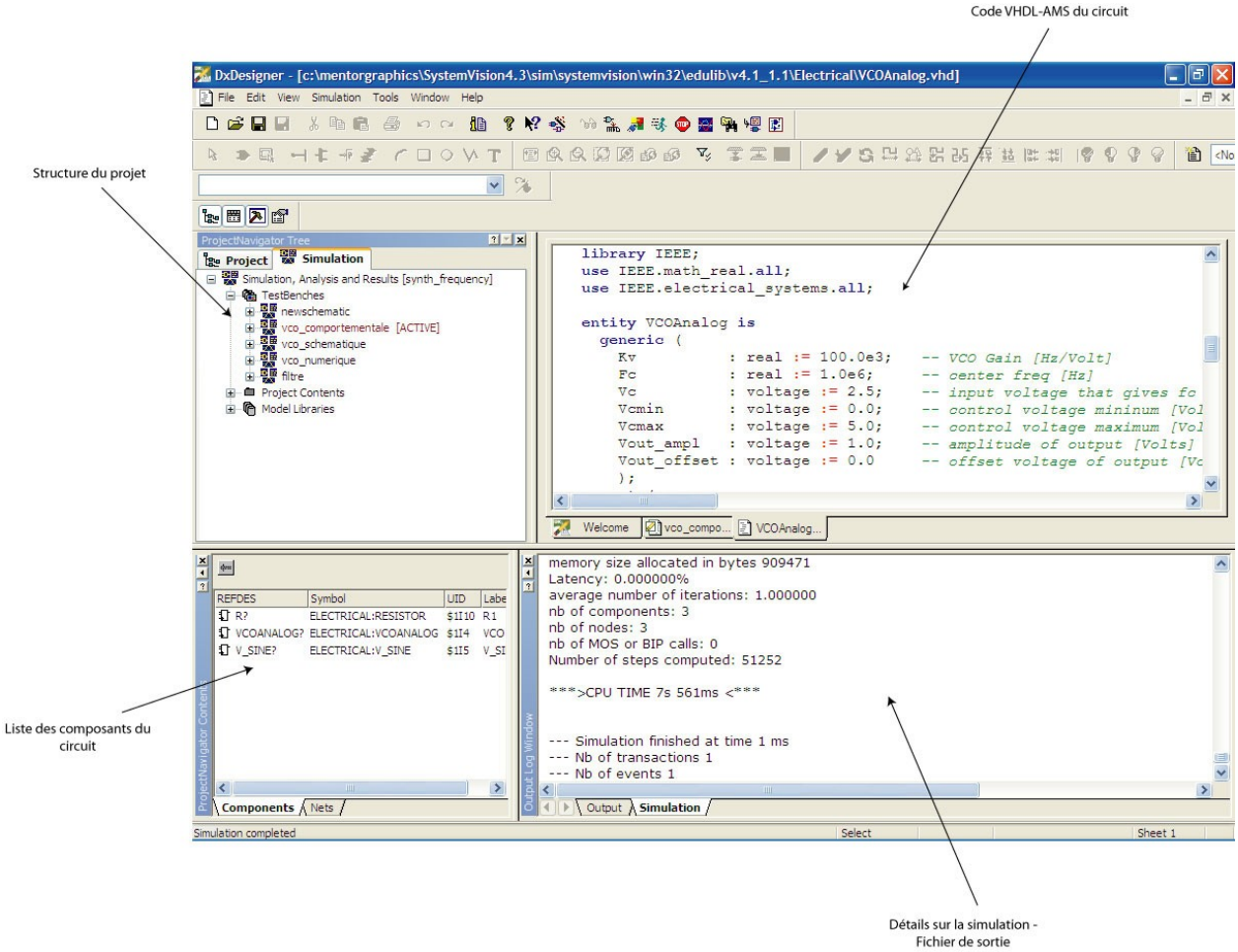


Figure 4.22 : Environnement de simulation de SYSTEM VISION

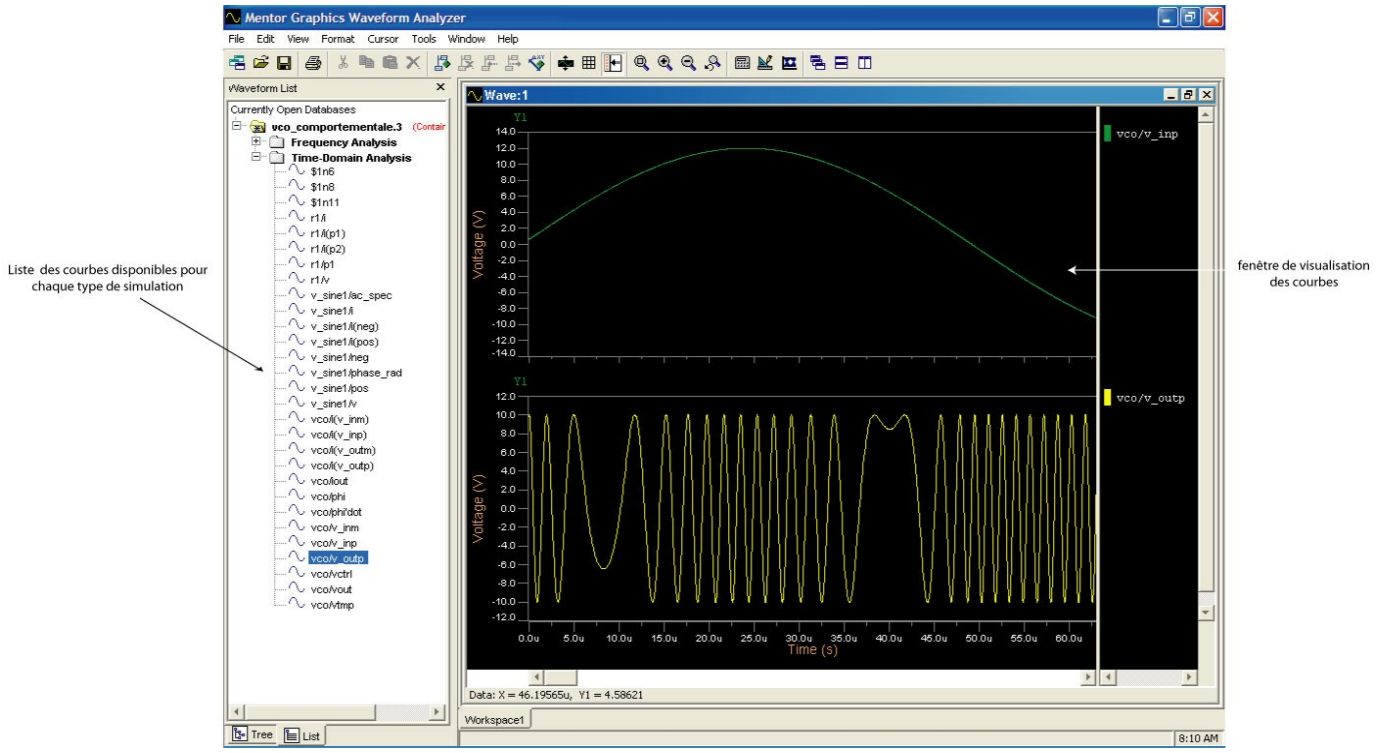


Figure 4.23 : Interface graphique de SYSTEM VISION pour la visualisation des courbes

3.2.2 Résultats de la simulation du synthétiseur de fréquence

La figure 4.24 illustre les réponses du modèle du synthétiseur fractionnaire. La première courbe montre l'allure du signal de référence à l'entrée de la boucle qui a la même allure et même fréquence que le signal à la sortie du diviseur fractionnaire. La deuxième illustre le courant à la sortie de l'accumulateur et la troisième courbe montre la variation de la période de sortie du VCO.

Les résultats obtenus prouvent le bon fonctionnement du synthétiseur et confirme la bonne modélisation de chaque bloc.

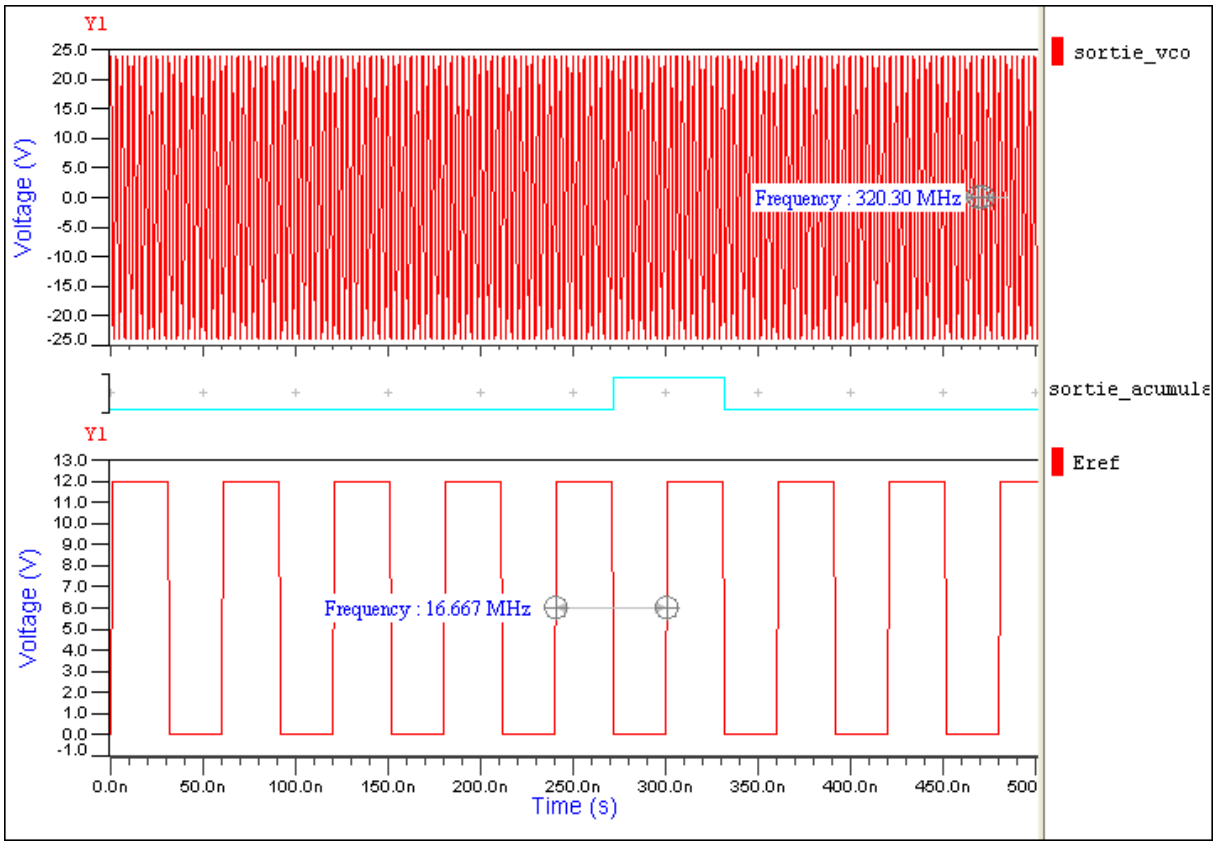


Figure 4.24 : Simulation du modèle du synthétiseur de fréquence

IV- Conclusion

Dans ce chapitre nous avons développés les modèles comportementaux pour l'application de synthèse de fréquence. Nous avons adoptées dans la majorité des cas l'approche fonctionnelle permettant au mieux l'exploitation des modèles de chaque blocs constituant le système pour d'autres applications.

Le modèle du synthétiseur fractionnaire a été simulé pour une utilisation répondant à la norme UMTS. Cette étape a permis de montrer la validité de chaque modèle.

L'étude de bruit de phase est importante pour les systèmes de télécommunications qu'on a simulé ici. En effet le bruit introduit des erreurs sur les signaux de sortie. La modélisation du bruit de phase se présente alors comme une perspective à l'issu de ce travail.

CONCLUSION

Le développement de langage de description matérielle associé aux outils de synthèse a largement contribué à l'automatisation de la conception des circuits.

Ainsi la simulation et la modélisation ont facilité les conceptions comme par exemple dans le domaine de l'électronique analogiques et numériques complexes. Les techniques de modélisation doivent ainsi être flexibles et capables de s'adapter au changement de la technologie et aux niveaux besoins.

L'utilisation de la modélisation comportementale répond à ces exigences. Le concepteur peut commencer par optimiser l'architecture et les paramètres génériques de son système en faisant des simulations comportementales avant de passer à des simulations niveau transistor de l'architecture optimisée. L'objet de la modélisation comportementale étant de décomposer le système en un ensemble de blocs fonctionnels, où chaque bloc ou certains d'entre eux sera remplacé par une description fonctionnelle et plus abstraite.

Un modèle comportemental fiable présente les caractéristiques suivantes :

- une description complète des caractéristiques de transfert du circuit niveau transistor ;
- une bonne précision par rapport au circuit réel
- convergence presque sûre des équations dans la modélisation pour les différentes conditions d'opérations et les différents modes de simulations ;
- paramètres génériques permettant d'adapter le modèle pour toute une classe de circuits similaires ;
- un gain de temps en simulation comportementale suffisamment important.

Quelques points techniques sur la conception d'un système ont été présenté dans ce mémoire afin d'introduire l'importance de la modélisation comportementale. Pour la modélisation du synthétiseur de fréquences, les modèles des différents blocs constituant le système qui ont été élaborés sont:

- le comparateur phase fréquence ;
- la pompe de charge ;
- le filtre passe-bas ;
- l'accumulateur et le diviseurN/M constituant le diviseur fractionnaire ;
- l'oscillateur contrôlé en tension (VCO).

La simulation du système codé en VHDL-AMS qui utilise la bibliothèque *« library IEEE »* a été réalisé sous l'environnement SYSTEMVISION de Mentor Graphics. Les paramètres choisis pour cette simulation répondent à la norme UMTS.

Un point complémentaire qu'on n'a pas réalisé dans ce travail et qui reste à étudier concerne la modélisation des bruits de phase pour le synthétiseur de fréquence.

Compte tenu de l'importance et de la capacité du langage VHDL-AMS plusieurs perspectives autour du thème qu'on a étudié ici se présentent. Nous pouvons proposés entre

autres le développement d'autres bibliothèques de modèles de systèmes électromécanique, thermique, hydraulique, colorimétrique ainsi que de la combinaison de ces différentes technologies.

ANNEXE 1 : SYSTEM VISION Quick reference guide [9]

SystemVision: Overview and Key Capabilities

SystemVision is a mixed-signal, mixed-technology simulation tool. It combines graphical design, waveform viewing, and HDL simulation technologies into one versatile package. You can use SystemVision as a "virtual lab" for exploring the design of complete multi-discipline systems, from concept to manufacturing.

Some key capabilities of SystemVision:

- You can simulate strictly VHDL-AMS models, strictly SPICE models, or combinations of both. The models themselves can be a mix of SPICE subcircuits and VHDL-AMS language descriptions.
- SystemVision includes a VHDL-AMS model library, with representative devices and effects spanning several technologies. Electrical (analog/digital), mechanical, hydraulic, magnetic, and thermal models are included.
- SystemVision can automatically generate symbols for user-created models and VHDL-AMS templates from user-created symbols.
- Hierarchical design methodologies and multi-page schematics are supported.
- Full mixed-signal design is supported, including buses.
- Designs are organized and managed as projects and simulated as testbenches.
- Each unique collection of simulation settings on a given testbench is saved as an experiment.
- It displays graphical results in an easy-to-use waveform viewer, which is launched in a separate window.
- It provides DC and parametric sweeps, Monte Carlo simulation, and sensitivity analysis.
- All initialization, configuration, and command files are generated as text (ASCII), which allows you to open, read, and edit them.

Basic Usage

Design Creation

Create a New Project

1. Select **File** > **Open** > **Project…**
2. Type in a Project name.
3. Click the Browse... button and navigate to a location where you want to locate the project.

Create a New Schematic (Design Root)

1. From within a project, select **File** > **New…**
2. Select Type **Schematic**.
3. Type in a Schematic name.
4. Click the Browse... button and navigate to a location where you want to locate the schematic.

Add Component Symbols to a Schematic

1. Choose **Add** > **Components...** from the main menu.
2. Select symbol library under Directory (left pane).
3. Select the symbol you want to place under Symbol (middle pane).
4. Click the Place button.
5. Move the mouse cursor onto the schematic and click to place symbol in the work area.

Specify Parameter Values for VHDL-AMS Components

1. Right-click on a component and choose **Edit Model Properties.**
2. Click the **Parameters** tab.
3. Under the Value column, type in a value for a parameter listed under the Name column.

Specify Parameter Values for SPICE Components

1. Double-click on a component or right-click on a component and choose **Properties.**
2. Click the **Attributes** tab.
3. In the Attributes pane, select the name of the SPICE parameter (which is displayed in the Name field).
4. In the Value field, type in the value.

Connect Component Symbols on the Schematic

1. Click **Net** button from the toolbar, or choose **Add** > **Net.**
2. Move cursor to component pin on schematic, then click-and-drag to create net (wire).
3. Release left mouse button to end wire.
4. Select a wire, then click-drag-release to make connections between component

5. Click OK to run the simulation.
6. Check the output window for any Warnings/Errors. The waveform viewer will not launch until errors are corrected.
7. If there are no errors, the waveform viewer launches automatically, and you can view results.

Viewing Results

View Waveforms

1. In the Waveform list, double-click on a waveform name to add it to a new region in the graph window.
2. *Overlay* multiple waveforms—With a waveform already in the graph window, select a waveform name in the Waveform list and drag it onto the displayed waveform. This overlays both waveforms together in the same graph.
3. *Stack* multiple waveforms—With a waveform already in the graph window, double-click a waveform name in the Waveform list. This stacks the waveforms in separate graphs.

Perform Measurement (between cursors).

1. Choose **Cursor** > **Add** (or press F5) twice.
2. Click and drag each cursor to desired location on waveform(s).
3. Click on the Measurement icon in the toolbar, which displays the Measurement Tool dialog box.
4. Specify settings for Measurement, Source Waveform(s), Measurement Setup,
5. For Measurement Results, select **Annotate Waveform(s) with Result Markers**.
6. For Apply Measurements to, select **Between Two Cursors**.
7. Click **Apply**. The measurement information appears in the graph area.

Outils équivalents

- ADVance-MS de Mentor Graphics (obtenu par l'intermédiaire du CNFM), l'un des premiers produits disponibles.
- D'autres outils gratuits fonctionnant sous Windows (et donc plus facilement utilisables de façon personnelle) ont été mis sur le marché depuis : hAMSter (Simec).

Annexe 2 : VHDL –AMS quick reference guide [10]

Syntax and Structure Overview

VHDL-AMS is an extremely powerful and versatile modeling language. Important aspects of the language are reviewed next.

Model entity structure

The structure of a VHDL-AMS model entity varies slightly depending on what type(s) of ports are used. The following example summarizes these variations:

```
entity entity_name is
 port (
  terminal port_name(s) : port_nature;                -- terminal port
  quantity port_name(s) : port_direction port_type;  -- quantity port
  signal port_name(s) : port_direction port_type      -- signal port
  );
```

- Terminal ports are bidirectional. The port_nature of a terminal port refers to its "technology type", such as electrical, fluidic, rotational, and so forth. For example,

  ```
  terminal port_name : fluidic;     -- terminal port
  ```

  Terminal ports are conservative, which means that they have effort (across) and flow (through) aspects associated with them, and they obey energy conservation laws (i.e. Kirchoff's laws and Newton's laws).

- Quantity ports are uni-directional, so the direction of the port must be specified in its declaration. These ports can be of type in or out. The port_type of a quantity port refers to the type of the port (i.e. real, integer, Boolean, and so forth.). For example, an output quantity port would be declared as type real using the following syntax:

```
quantity port_name : out real;  -- quantity port of type real
```

  Quantity ports are not conservative, which means that they do not have effort (across) and flow (through) aspects associated with them, and they do not obey energy conservation laws. However, for model solvability, one equation must be added to a model for each quantity of type out.

- Signal ports can be bi-directional, but work in only one direction at a time. These ports are typically declared as either in or out, but can also be declared as inout. The port_type of signal ports refers to the type of the port (i.e. real, integer, Boolean, and so forth.). For example, an output signal port would be declared as type integer using the following syntax:

```
signal port_name : out integer;  -- signal port of type integer
```

  As mentioned previously, the inclusion of the "signal" identifier on signal ports is optional. For example, the following two port declarations are equivalent:

```
signal port_name : in std_logic;   -- signal port
```
and
```
port_name : in std_logic;  -- signal port
```

  Signal ports are not conservative, which means that they do not have effort (across) and flow (through) aspects associated with them, and they do not obey energy conservation laws.

Model architecture structure

The structure of a VHDL-AMS model architecture is illustrated as follows:

```
architecture architecture_name of entity_name is
  internal object declarations
begin
  concurrent statement(s);  -- device equations
end architecture architecture_name;
```

The behavior of a VHDL-AMS model is described with one or more concurrent statements located between the **begin** and **end architecture** keywords.

Libraries and use clauses

Object types and other information are declared in packages. These packages are located in libraries and are accessed with the **use** clause:

```
library IEEE;
use IEEE.electrical_systems.all;
```

These lines will typically be included in all electrical models. To access all of the std_logic_1164 digital packages declared in the IEEE library, the following line should be added:

```
library IEEE;
use IEEE.electrical_systems.all;
use IEEE.std_logic_1164.all;
```

Syntax specifics

- Comments can be inserted in VHDL-AMS models by using two dashes "--" followed be the comment. For example:
  ```
  a == b*c;     -- everything to the right of the dashes
                -- (to the end of the line) is a comment.
  -- even otherwise reserved words like port and architecture
  -- can be included in comments.
  ```

- White space characters (like tabs, spaces, and carriage returns) are ignored unless specifically required for language element separation.

- Indents are optional, but are often used to enhance legibility.

- VHDL-AMS statements are terminated with a semi-colon ";". Statements without a semicolon may span multiple lines. For example:

```
port (port_type port_name(s) : port_nature); -- one line statement
```

can also be written as

```
port (                                    -- statement begins
  port_type port_name(s) : port_nature    -- statement continues
  );                                      -- statement ends
```

- Elements in a list are separated by commas ",". For example:

```
(element1, element2, element3,...)
```

- Commas (and colons) do not require spaces.

- Identifiers are names modeler's give to objects they declare. The following restrictions apply to identifiers:

  - Must begin with a letter (A(a) – Z(z)).
  - May contain letters, numbers, and underscores. Identifiers may not begin or end with an underscore. Two or more underscores may not appear together.
  - Are not case sensitive.
  - Cannot be reserved words (**keywords**).

Literals

Literals are lexical (text) elements that represent immediate data values. There are four kinds of literals in VHDL-AMS:

- Number – there are two kinds of numbers in VHDL-AMS, integer literals and real literals. Either type can use exponential notation:

  - Integer literal examples: 7, 5e-6 -- must *not* contain decimal point.

  - real literal examples: 7.0, 5.0e-6 -- *must* contain decimal point

- Character – character literals are enclosed in single equation marks: '0', 'A'.

- String – string literals are enclosed in double quotation marks: "string literal example".

- Bit String – bit string literals are enclosed in double quotation marks: "01100010".

Selected Operators

VHDL-AMS supports several standard operators. Commonly used operators are summarized in this section.

Relational operators

Operator	Meaning
=	equal
/=	not equal
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Table 2 - Relational Operators

Arithmetic operators

Operator	Meaning
+	addition
-	subtraction
*	multiplication

/	division
**	exponentiation
mod	modulo
rem	remainder
abs	absolute value

Table 3 - Arithmetic Operators

Logical operators

Operator	Meaning
and	logical and
or	logical or
nand	negative (not) and
nor	negative (not) or
xor	exclusive or
xnor	exclusive-nor
not	logical not

Table 4 - Logical Operators

Concatenation operator

Operator	Meaning
&	concatenation

Table 5 - Concatenation operator.

Object Types

VHDL-AMS uses six classes of objects as shown in Table 6.

Object	Meaning
Constant	named literal value, non-changing during simulation
Terminal	analog port
Quantity	analog valued object
Variable	discrete valued object
Signal	digital port and discrete valued object
File	data storage object

Table 6 - Object types.

Commonly Used Predefined Attributes[14]

Selected attributes of scalar types

Attribute	Result
T'left	leftmost value in T
T'right	rightmost value in T
T'low	least value in T
T'high	greatest value in T
T'ascending	true if T is ascending in range, but false otherwise
T'image(x)	a textual representation of the value x of type T
T'value(s)	value in T represented by the string s
T'pos(x)	position number of x in T
T'val(x)	value at position x in T

Table 7 - Attributes of scalar types.

Selected attributes of signals

Attribute	Result
S'delayed(t)	signal with the same value of S, but delayed by time t (t >= 0)
S'transaction	signal changes value in simulation cycles in which a transaction occurs on S
S'event	true if an event has occurred on S in the current simulation cycle, false if otherwise
S'ramp (trise, tfall)	quantity that follows corresponding signal S with a linear change of value governed by trise (rising change) and tfall (falling change)
S'slew (rslope, fslope)	quantity that follows signal S with a linear change of value governed by rslope (rising slope) and fslope (falling slope)

Table 8 - Attributes of signals.

Selected attributes of quantities

Attribute	Result
Q'dot	derivative with respect to time of Q

Q'integ	time integral of Q from time 0
Q'delayed(t)	quantity equal to Q but delayed by t
Q'above(E)	signal that is true if Q >E, false if otherwise
Q'slew(rslope, fslope)	quantity that follows signal Q but with its derivative limited by rslope (rising slope) and fslope (falling slope)
Q'ltf(num, den)	Laplace transfer function of Q with num as the numerator and den as the denominator polynomial coefficients

Table 9 - Attributes of quantities.

Attributes of array types

Attribute	Result
A'left(n)	leftmost value in index range of dimension n
A'right(n)	rightmost value in index range of dimension n
A'low(n)	least value in index range of dimension n
A'high(n)	greatest value in index range of dimension n
A'range(n)	index range of dimension n
A'reverse_range(n)	index range of dimension n reversed in direction and bounds
A'length(n)	length of index range of dimension n
A'ascending(n)	true if index range of dimension n is ascending, false otherwise

Table 10 - Attributes of array types.

Assignment Statements and Simultaneous Equation Sign

Syntax	Meaning
<=	Signal assignment statement
:=	Variable assignment statement
==	Sign used in simultaneous equations

Sequential Statements

The following statements are sequential, and only appear in VHDL-AMS processes and subprograms:

Sequential if statements

If statement basic syntax

```
if boolean_expression then
  sequential statement(s);
elsif boolean_expression then
  sequential statement(s);
else
  sequential statement(s);
end if;
```

If statement example

```
if addr1 = '1' then
  a <= b;  -- signal assignment
  w := x;  -- variable assignment
elsif addr1 = '0' then
  a <= c;
  w := y;
else -- if addr1 is 'Z', 'X', etc.
  a <= d;
  w := z;
end if;
```

Sequential case statement

Case statement basic syntax

```
case expression is  -- expression includes series of alternative choices
  when choices =>  -- which can be tested with keyword when
    sequential statement(s);
end case;
```

Case statement model example

```
case switch is  -- expression identifier switch
  when on =>   -- consists of enumerated value "on"…
    sw_resistance <= r_on;
  when off =>   -- and enumerated value "off"
    sw_resistance <= r_off;
end case;
```

Sequential loop statements

Loop statement basic syntax

```
loop
  sequential statement(s);
end loop;

while boolean_expression loop
  sequential statement(s);
end loop;

for identifier in discrete_range loop
  sequential statement(s);
end loop;
```

For loop statement example

```
for count in 1 to 10 loop
  count_total := count_total + count;
end loop;
```

Simultaneous Statements

The following statements are simultaneous, and appear in concurrent sections of a model:

Simultaneous if statements

If statement basic syntax

```
if boolean_expression use
  simultaneous equation(s);
elsif boolean_expression use
  simultaneous equation(s);
else
  simultaneous equation(s);
end use;
```

If statement example

```
if vin'above(limit_high) use
  vout == limit_high;
elsif not vin'above(limit_low) use
  vout == limit_low;
else  -- vin is within limit_high and limit_low range
  vout == k*vin;
end use;
```

Simultaneous case statement

Case statement basic syntax

```
case expression is  -- expression includes series of alternative choices
  when choices =>  -- which can be tested with keyword when
    simultaneous statement(s);
end case;
```

Case statement model example

```
case res_switch use  -- expression identifier res_switch
  when on =>   -- consists of enumerated value "on"…
    vout == iout*r_on;
  when off =>   -- and enumerated value "off"
    vout == iout*r_off;
end case;
```

Standard Library

Commonly-used real math functions from standard library

Function	Meaning
sign(x)	sign of x
ceil(x)	ceiling of x
floor(x)	floor of x
round(x)	x rounded to nearest integer value
trunc(x)	x truncated toward 0.0
sqrt(x)	square root of x
cbrt(x)	cubed root of x
"**"(n,y)	n to the y power
exp(x)	e t the x power
log(x)	natural log of x
log2(x)	log base 2 of x
log10(x)	log base 10 of x
log(x,BASE)	log base BASE of x
realmax(x,y)	returns algebraically larger of x and y
realmin(x,y)	returns algebraically smaller of x and y
mod(x,y)	modulus of x/y
sin(x)	sine of x (radians)
cos(x)	cosine of x (radians)
tan(x)	tangent of x (radians)
arcsin(x)	inverse sine of x
arccos(x)	inverse cosine of x
arctan(x)	inverse tangent of x
arctan(y,x)	inverse tangent of point (y, x)

sinh(x)	hyperbolic sine of X
cosh(x)	hyperbolic cosine of X
tanh(x)	hyperbolic tangent of X
arcsinh(x)	inverse hyperbolic sine of X
arccosh(x)	inverse hyperbolic cosine of X
arctanh(x)	inverse hyperbolic tangent of X

Table 11 - Standard library functions.

Commonly-used math constants from standard library

Function	Meaning
math_e	e
math_1_over_e	$1/e$
math_pi	π
math_2_pi	2π
math_1_over_pi	$1/\pi$
math_pi_over_2	$\pi/2$
math_pi_over_3	$\pi/3$
math_pi_over_4	$\pi/4$
math_3_pi_over_2	$3\pi/2$
math_log_of_2	ln 2
math_log_of_10	ln 10
math_log2_of_e	$\log_2 e$
math_log10_of_e	$\log_{10} e$

math_sqrt_2	$\sqrt{2}$
math_1_over_sqrt_2	$1/\sqrt{2}$
math_sqrt_pi	$\sqrt{\pi}$
math_deg_to_rad	$2\pi/360$
math_rad_to_deg	$360/2\pi$

Table 12 - Standard library math constants.

In addition to these real functions, the math_real library also defines a complete set of complex functions.[15]

Annexe 3: Calculs des paramètres du synthétiseur fractionnaire

1) *Les paramètres du diviseur fractionnaire [8]*

Pour le synthétiseur de fréquence fractionnaire, nous avons fixés :

F_{REF} = 100 MHz

Rapport de division du diviseur à l'entrée : M = 6

Fréquence de division :T

L'accumulateur attaque le diviseur N/M. L'expression du facteur de division non entier est donné par l'expression suivante :

$$N_{frac} = M\left(\frac{K}{T}\right) + N\left(\frac{T - K}{T}\right) = N + \frac{K(M - N)}{T}$$

On en déduit l'expression de la fréquence de sortie du VCO :

$$F_{VCO} = F_{REF}\,\frac{M \cdot K + N \cdot (T - K)}{T}$$

Pour synthétiser une fréquence égale à $F_{vco}+F_{STEP}$, on incrémente K par 1 et on aura :

$$F_{VCO} + F_{STEP} = F_{REF}\,\frac{M \cdot (K + 1) + N \cdot (T - K - 1)}{T}$$

On en déduit alors F_{STEP}

$$F_{STEP} = F_{REF}\,\frac{M - N}{T}$$

On en déduit l'expression de la fréquence de division T

$$T = F_{REF}\,\frac{M - N}{F_{STEP}}$$

Application numérique : $T = \frac{100}{5} = 20$

Rapport de division N/N+1

Suivant la norme UMTS , la bande d'émission / réception est compris entre 1920 MHz et 1980 MHz

$$N_{frac} = N + \frac{K(M - N)}{T}$$

En faisant varier K entre 1 et T, N_{frac} peut être compris entre les bornes suivantes :

$$\frac{1920}{100} \leq N_{frac} \leq \frac{1980}{100} \Leftrightarrow 19.2 \leq N_{frac} \leq 19.8$$

Compte tenu de ces bornes de N_{frac} , nous pouvons tirer la valeur du rapport de division.

$$N / N + 1 = 19 / 20$$

Valeur de l'entrée fractionnaire : K

Tenant compte de la valeur de la fréquence à synthétiser, nous pouvons calculer la valeur de K. Prenons l'exemple où la fréquence à synthétiser est égale à 1980 MHz , calculons la valeur de K.

$$N_{frac} = 19.5 = \frac{K.20 + (20 - K).19}{20}$$

d'où *K=10*

2) *Les paramètres du filtre passe bas [13]*

$$H(p) = H_o \frac{\omega_p^2}{p^2 + \frac{\omega_p}{Q_p} p + \omega_p^2}$$

Pour le filtre passe bas de la figure 4.7 au chapitre 4, on aura :

$$F_p = 117kHz$$

$$Q \approx 1$$

$$H_o \approx 1$$

Le courant de la pompe de charge $I_{pompe} = 100\mu A$

3) Les paramètres du VCO

Le gain du VCO $K_{vco} = 50MHz$

$F_{\max vco} = 350MHz$ et $F_{\min vco} = 300MHz$

REFERENCES

[1] Sabeur JEMMALI, « Contribution à l'Elaboration de Méthodologies et d'Outils d'Aide à la Conception de Systèmes Multi Technologiques », Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications Paris, Novembre 2003.

[2] « Introduction à VHDL », http://www.comelec.enst.fr/hdl/vhdl_intro.html

[3] Alain VACHOUX, « Modélisation de Systèmes Intégrés Analogiques et mixtes – Introduction à VHDL-AMS », Version 2002.

[4] François LEMERY, « Modélisation Comportementale des Circuits Analogiques et Mixtes », Thèse de Doctorat en Microélectronique, Institut National Polytechnique de Grenoble, Novembre 1995.

[5] Samia BELKACEM, « Macromodélisation comportementale de circuits analogiques : application au circuit convoyeur de courant », Mémoire de fin d'étude pour l'obtention du diplôme de Magister en Microéléctronique, Univeristé de Batna-Faculté des Sciences de l'Ingénieur Département Electronique, 2005.

[6] Olivier SUSPLUGAS, « Application des Boucles à Verrouillage de Retard à la Synthèse de Fréquences dans les Circuits pour les Communication Mobiles », Thèse de Doctorat , Ecole Nationale Supérieure des Télécommunications de Paris, Juin 2003.

[7] http://vhdl.org

[8] Ahmed FAKHFAKH, « Contribution à la Modélisation Comportementale des Circuits Radio Fréquence », Thèse de Doctorat en Electronique, Université de Bordeaux I, Janvier 2002.

[9] Mentor Graphics, « SystemVision Quick Reference Guide » – SystemVision v4.3 – February 2006.

[10] Scott Cooper, Mike Donnelly, Darell Teegarden , « How to model mechatronic system using VHDL-AMS », SystemVision™ Technology Series - Mentor Graphics.

[11] S. Jemmali, A. Nehme, and J. J. Charlot, « Behavioral modeling of multitechnological systems with VHDL-AMS and simulating with spice », Proceedings of the 2003 International Workshop on Behavioral Modeling and Simulation, BMAS 2003, pages: 92 – 96, 7-8 Oct. 2003.

[12] M. Zorzi; F. Franze; N. Speciale, and G. Masetti; « A tool for the integration of new VHDL-AMS models in SPICE », Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS 04, pages: IV - 637- 40 Vol.4, 23-26 May 2004.

[13] Paul Bildstein, «Synthèse et réalisation des filtres actifs», Technique de l'Ingénieur, traité Electronique E 3 130

Auteur : RAMANANTSIHOARANA Harisoa Nathalie

Titre : MODELISATION COMPORTEMENTALE PAR VHDL-AMS D'UN SYNTHETISEUR DE FREQUENCE

Nombre de pages : 83
Nombre de tableaux : 10
Nombre de figures : 39

Résumé :

Ce travail présente l'étude de la modélisation comportementale de circuit mixte : analogique et numérique avec le langage de description matérielle VHDL-AMS.

Le langage VHDL-AMS et ses instructions sont présentés en faisant ressortir leurs fonctionnalités. Quelques méthodologies de modélisation comportementale sont étudiées.

Le modèle comportemental du synthétiseur de fréquence, représentatif d'un système mixte, est développé selon une approche fonctionnelle. Pour la validation du modèle les paramètres du synthétiseur sont choisis suivant la norme UMTS pour la simulation.

Mots-clés :

VHDL-AMS, Modélisation, Modélisation comportementale, Synthétiseur de fréquence, Langage de description matérielle.

Directeur de mémoire :

M. RASTEFANO Elisée

Adresse de l'auteur :

LOT VB 72 TER AC Ambatoroka -101- Antananarivo