N° d'ordre : 2411

THESE

PRESENTEE A

L'UNIVERSITE BORDEAUX I

ECOLE DOCTORALE DES SCIENCES PHYSIQUES ET DE L'INGENIEUR

Par **Ahmed FAKHFAKH**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPECIALITE : Electronique

CONTRIBUTION A LA MODELISATION COMPORTEMENTALE DES CIRCUITS RADIO-FREQUENCE

Soutenue le : **11 Janvier 2002**

Après avis de :

MM.	**L. KAMOUN**	*Professeur*	*ENIS - Sfax – Tunisie*	**Rapporteurs**
	J.J. CHARLOT	*Professeur*	*ENST – Paris – France*	

Devant la commission d'examen formée de :

MM.	**G. CAMBON**	*Professeur*	*LIRMM – Montpellier*	**Président**
	N. LEWIS	*Maître de Conférences*	*IXL – Université Bordeaux 1*	**Rapporteur**
	J.J. CHARLOT	*Professeur*	*ENST – Paris*	**Examinateurs**
	P. FOUILLAT	*Professeur*	*IXL – Université Bordeaux 1*	
	L. KAMOUN	*Professeur*	*ENIS - Sfax*	
	H. LEVI	*Professeur*	*IXL – Université Bordeaux 1*	
	J.P. MORIN	*Ingénieur*	*ST Microelectronics - Crolles*	
	M. ROBBE	*Ingénieur*	*EADS - Bois d'Arcy*	

- 2002 -

A ma famille

Remerciements

Les travaux de recherche présentés dans ce mémoire ont été réalisés au Laboratoire de Microélectronique IXL de l'Université Bordeaux I, dirigé par Monsieur le Professeur André TOUBOUL.

Je tiens tout d'abord à remercier Monsieur André TOUBOUL de m'avoir accueilli dans le laboratoire IXL et pour l'importance qu'il donne à la coopération entre l'IXL et l'ENIS (Tunisie).

Mes remerciements vont aussi aux messieurs Lotfi KAMOUN et Jean-Jacques CHARLOT, les deux rapporteurs de thèse, pour leur lecture attentive et critique du mémoire.

Je remercie également Monsieur Hervé LEVI, le directeur de la thèse, pour la confiance qu'il m'a accordée durant les années de recherche, pour son amitié et son respect.

Mes remerciements vont également à Madame Milet LEWIS pour sa précieuse contribution et son soutien constant.

Je remercie Monsieur Pascal FOUILLAT, le directeur de l'axe de recherche Conception et Test pour son soutien et sa présence dans le jury de thèse.

Je remercie vivement Monsieur Gaston COMBON pour avoir présidé le jury de thèse.

Mes remerciements pour Messieurs Jean-Paul MORIN et Michel ROBBE pour l'intérêt qu'ils portent à ces travaux en acceptant de faire partie du jury de thèse.

Je tiens à remercier les membres des équipes de l'axe de recherche Conception et Test pour leur soutien, en particuliers Messieurs Franck BADETS, Yann DEVAL, Jean-Baptiste BEGUERET et Mademoiselle Anne SPATORO de l'équipe Conception ; Thomas ZIMMER, Bertrand ARDOUIN et Hassène MNIF de l'équipe Caractérisation et Modélisation.

Que Messieurs Régis DEVRESSE et Partick VILLESUZANNE reçoivent mes remerciements pour le support informatique qu'ils assurent.

Je tiens également à remercier Messieurs François MARC et Didier GEOFFROY pour leur disponibilité, leur gentillesse et leur amitié.

Enfin, ce travail de thèse n'aurait pas été mené à bien sans la présence et la disponibilité de nombreuses personnes : merci à tous les membres du laboratoire IXL.

Table des matières

Introduction générale

L'industrie électronique représente de nos jours la première industrie. Le marché du téléphone portable par exemple atteindra en 2005 un chiffre d'affaire de 100 milliards d'Euros, soit le tiers du budget de la France.

L'évolution des systèmes électroniques se fait à plusieurs niveaux : la miniaturisation, la fiabilité, la réduction du coût et la réduction de la consommation. Pour répondre à tous ces objectifs, les nouvelles tendances en électronique sont l'intégration des System on Chip (SoC) et des ASIC (Application Specific Integrated Circuit) mixtes, c'est à dire intégrant des fonctions analogiques et mixtes.

Le coût de la production baisse avec la miniaturisation mais il est aussi fonction du coût de la conception. Les outils de conception assistée par ordinateur (CAO) doivent évoluer pour suivre l'évolution technologique de l'industrie électronique. Dans un environnement CAO, on dispose de divers outils pour passer du concept à la description physique du circuit. Ces outils regroupent les interfaces de saisie de schémas, les simulateurs, les interfaces graphiques, les bibliothèques de modèles, les outils de dessin des masques et de vérification, etc.

Les simulateurs, apparus dans les années 1970, ont évolué au cours des trente dernières années. On en distingue actuellement trois types : les simulateurs numériques, les simulateurs analogiques et les simulateurs mixtes. La dernière génération des simulateurs mixtes est basée sur l'utilisation d'un cœur de simulation unique permettant de simuler simultanément les parties analogiques et numériques décrites dans un même langage.

Pour simuler un circuit avant qu'il ne parte en fabrication, il existe plusieurs niveaux de description. Pour les circuits analogiques, le premier niveau est appelé niveau transistor puisque les modèles élémentaires considérés sont les modèles des transistors, des résistances ou des capacités, etc. Pour les circuits numériques, il est appelé niveau porte logique puisqu'il s'agit de modèles de portes logiques, de bascules, etc. L'évolution des outils CAO a fait naître un niveau de description plus élevé qui est le niveau comportemental. Il ne s'agit plus de décrire le circuit avec les éléments électroniques de base, mais de décrire son comportement électrique externe à l'aide d'un langage adéquat. Sont apparus alors des langages standards tel que VHDL et Verilog pour les circuits numériques et VerilogA et récemment VHDL-AMS pour les circuits analogiques et mixtes.

La description comportementale à l'aide d'un langage standard est devenu aujourd'hui une étape incontournable dans la conception de circuits numériques. Elle est utilisée dans toutes les phases de la conception : de la spécification à la réalisation, en passant par la synthèse numérique. Cette démarche est une démarche hiérarchique descendante, appelée aussi **Top-Down**. Elle part d'une description fonctionnelle du système pour aboutir automatiquement à son architecture au niveau porte logique à l'aide des outils de synthèse.

La nouvelle norme VHDL-AMS, apparue en 1999, permet l'application de la démarche hiérarchique Top-Down pour les circuits analogiques et mixtes. Cependant, on ne dispose pas encore sur le marché d'outil de synthèse analogique et mixte et le chemin paraît encore long pour acquérir un tel outil.

Les travaux présentés dans ce mémoire apportent une contribution à la modélisation comportementale des circuits analogiques et mixtes en termes de méthodologie et de développement de bibliothèques de modèles standards, indispensables à l'évolution du flot de conception analogique Top-Down.

Le premier chapitre est une description du contexte de l'étude. Il propose une introduction sur les défis de l'électronique moderne et présente les outils et méthodes pour la conception des circuits analogiques et mixtes. De cette réflexion apparaissent les nouveaux besoins pour le développement des outils CAO.

Le chapitre 2 s'intéresse aux méthodes de modélisation comportementale. En effet, après l'apparition de la norme VHDL-AMS, la question à laquelle il faut trouver une réponse est quelle méthode ou démarche peut-on adopter pour développer des modèles comportementaux fiables ? Dans ce chapitre, nous développons deux approches de modélisation : une approche schématique et une approche fonctionnelle. Notre objectif est de transmettre un savoir-faire et les méthodes proposées peuvent aider les concepteurs à développer leurs propres modèles en un délai raisonnable.

Notre souci était aussi de pouvoir dégager une démarche systématique de modélisation qui servira de base au développement d'outils d'automatisation de la modélisation.

Une deuxième étape indispensable à l'évolution du flot de conception analogique et mixte est le développement de bibliothèques de modèles standards. Le chapitre 3 expose le développement d'une bibliothèque comportementale pour les circuits radio-fréquence (RF). Nous commençons par décrire la bibliothèque de modèles en expliquant à chaque fois le principe de fonctionnement de chaque circuit modélisé et en détaillant le principe de la modélisation. La bibliothèque développée est dédiée aux applications de synthèse de fréquence.

Pour valider les modèles développés, nous avons comparé leurs performances aux simulations niveau transistor et aux mesures faites sur des prototypes fabriqués. Il s'agit de trois applications.

La première application consiste à modéliser un oscillateur synchrone 2.4 GHz de type COLPITTS dédié aux réseaux locaux sans fil. Le prototype fabriqué a été intégré dans la technologie BiCMOS AMS 0.8 µm. La deuxième application consiste à modéliser un oscillateur synchrone 5.2 GHz à résistance négative, dédié aux applications de type HIPERLAN. Le prototype fabriqué a été intégré dans la technologie HCMOS (CMOS 0.25 µm) de STmicroelectronics. La dernière application est la modélisation d'un synthétiseur de fréquence fractionnaire répondant à la norme UMTS'2000. Le gain considérable en temps de

simulation du synthétiseur montre l'apport de la modélisation comportementale et promet des avantages pour optimiser la conception de ce genre d'applications.

Lors du développement de la bibliothèque RF, un point précis a attiré notre attention. En effet, un des critères important lors de la conception des circuits RF est l'optimisation du bruit de phase. Dans le dernier chapitre de ce mémoire, le chapitre 4, nous abordons le problème du bruit de phase dans les oscillateurs.

Nous commençons par des généralités sur les signaux aléatoires en rappelant les différentes définitions et propriétés de ce genre de signaux. Ceci est indispensable pour manipuler les grandeurs définissant le bruit de phase.

Dans un second paragraphe, nous étudions le bruit de phase dans les oscillateurs en rappelant ses caractéristiques dans les domaines fréquentiel et temporel. Nous développons également les relations de passage entre les grandeurs fréquentielles et temporelles.

La deuxième partie du chapitre sera consacrée à la présentation de notre méthode de modélisation comportementale du bruit de phase, pour tenir compte de l'effet de cette perturbation dans nos modèles d'oscillateurs. Pour valider la méthode, nous nous sommes intéressés à la modélisation comportementale du bruit de phase dans les oscillateurs synchrones. Nous avons alors comparé les caractéristiques de bruit de phase obtenues avec des simulations comportementales à celle mesurées avec des analyseurs de spectre sur le prototype fabriqué. Les résultats obtenus nous encouragent à étendre notre méthode de modélisation aux autres types de circuits constituant les systèmes de synthèse de fréquence, tels que les comparateurs de phase et les diviseurs de fréquence.

Chapitre 1

Conception des circuits analogiques et mixtes : les nouvelles tendances

I Introduction : les défis de l'électronique moderne

I.1 Le marché de l'électronique

En ce début du 21ème siècle, l'électronique a envahi la vie quotidienne et l'industrie qui en est issue est restée la première industrie à l'échelle mondiale. Les progrès concernant les techniques de fabrication mais aussi les outils de CAO (Conception Assistée par Ordinateur) ont permis d'atteindre une production de masse à un coût attractif. La loi de Moore, établie empiriquement, stipule qu'il faut investir au moins 10% du chiffre d'affaires en recherche et développement. Ainsi, plus on vend, plus on investit pour concevoir de nouveaux produits qui permettront de détenir de nouvelles parts du marché. Il ne s'agit plus alors de suivre une demande mais de l'anticiper en créant un besoin ou en imaginant des applications.

Le coût de production baisse en fonction de l'augmentation du nombre de circuits obtenus à partir d'une même tranche de silicium. Ainsi, une décroissance annuelle de prix de marché de 25% implique une réduction de la surface du silicium par deux tous les deux ans [RKL01]. Partant de ce principe, de gros efforts de recherche ont été investis pour la diminution de la largeur des canaux des transistors qui approche de nos jours 100 nm en production industrielle. Ainsi, un Pentium 4 par exemple contient 42 millions de transistors et fonctionne à plus de 1 GHz [OUD00]. Aux environs de 2010, on aura plus d'un milliard de transistors sur une puce de silicium [RKL01].

Le coût de production est aussi fonction du coût de la conception. Les outils de CAO doivent suivrent l'évolution de la productivité technologique ; cependant, un écart a été creusé et les concepteurs doivent rattraper leur retard. Les outils CAO évoluent maintenant vers les couches abstraites et surtout les réponses systèmes avec la réutilisation massive de blocs IP *(Intellectual Property)* [RKL01]. L'IP ou '*composant virtuel'* est un module représentant une fonction électronique qui peut être rapidement utilisé pour être inséré dans un circuit intégré. Aujourd'hui, sur un circuit de 30 millions de portes, nous n'en créons réellement qu'une centaine de milliers et pour les circuits du futur, la part de nouveauté représentera 1.5 millions de portes sur un circuit qui en comptera au moins dix fois plus. L'IP est, sans conteste, l'affaire qui devrait faire réaliser un bond substantiel à la productivité des concepteurs. *'Demain, il y aura probablement des concepteurs de boîtes et des assembleurs de boîtes'*

prévoit le professeur Michel Robert du laboratoire d'informatique de robotique et de microélectronique de Montpellier (LIRMM) [RKL01].

I.2 La part et l'évolution des circuits analogiques

L'évolution des systèmes électroniques se fait à plusieurs niveaux et même si les chercheurs travaillent actuellement sur l'électronique d'après demain, basée sur des principes de fonctionnement différents de ceux utilisés à l'heure actuelle (principes quantiques, électronique mono-électron, électronique moléculaire, …), les architectures classiques basées sur les transistors MOS ou Bipolaires sont encore à l'ordre du jour [HER01]. Cependant, ces transistors présentent des effets nouveaux ou rendent importants des effets qui étaient secondaires voire négligeables dans les anciennes technologies. Il est donc nécessaire de tenir compte de tous ces effets dans les modèles de transistors pour assurer une conception de qualité.

Dans la plupart des systèmes électroniques, le traitement du signal est réalisé après numérisation, principalement pour des raisons de précision et d'immunité au bruit. Cependant, la nature analogique et le temps continu des signaux physiques rendent impossibles l'élimination complète des circuits analogiques. Beaucoup de systèmes intégrés sont donc des circuits mixtes analogique-numérique. Généralement, la partie analogique constitue 10% de la surface du circuit.

Le marché d'aujourd'hui est motivé par la miniaturisation, la fiabilité, la réduction du coût et la réduction de la consommation. Ces défis font que les circuits analogiques sont en train de gagner du terrain. En effet, leurs performances en terme de rapidité, de surface occupée et de consommation font qu'ils deviennent indispensables dans la conception de certains systèmes devant les limites de performances des circuits numériques. En 2001, le pourcentage des circuits mixtes est de 25% par rapport aux circuits électroniques. Ce pourcentage passera à 70% en 2006, d'après les prévisions de l'association 'The semiconductor Industry Association' à San José, USA [MAR01].

Une nouvelle tendance consiste aussi à évoluer vers les systèmes micro-électromécaniques qui incluent des capteurs (température, pression, champ magnétique, …) et des micro-actionneurs (moteurs, vérins, miroirs pivotants, …) associés à l'électronique de contrôle. On les trouve de plus en plus dans beaucoup de secteurs d'application tels que le

biomédical, l'espace, l'industrie chimique, l'environnement ou les télécommunications. Ces micro-systèmes, ou MEMS (Micro Electro Mechanical Systems) sont en pleine évolution et font cohabiter des circuits numériques et analogiques dans des ASICs mixtes.

Pour suivre cette évolution, les outils CAO doivent nécessairement s'adapter. Les nouveaux besoins peuvent se résumer en les points suivants :

- Développement de modèles de transistors de plus en plus complexes tenant compte des nouveaux effets liés au développement des technologies de fabrication
- Densité d'intégration de plus en plus élevée entraînant des difficultés de simulation et nécessitant de nouvelles techniques de résolution mathématique
- Nécessité de développer les outils de simulation des circuits mixtes. D'une part, il ne suffit plus de concevoir des blocs numériques et analogiques séparément mais il faut s'assurer que l'ensemble du système mixte fonctionne correctement dans le même environnement de simulation. D'autre part, la conception des circuits analogiques, contrairement aux circuits numériques, n'est pas encore automatisée ce qui augmente le temps de conception donc le coût
- Développement d'outils de simulation des micro-systèmes

A tout cela viennent s'ajouter des contraintes de conception industrielles fortes : conformité au cahier des charges, détection rapide des erreurs dans le cycle de conception, fiabilité du circuit fabriqué, gestion de la durée des projets pour satisfaire le « time to market », réutilisabilité des conceptions, gestion de grosses équipes de conception, simulation d'un circuit en tenant compte de son environnement, etc.

Pour tenir compte de toutes ces contraintes, la méthodologie de conception utilisée joue et jouera un rôle de plus en plus primordial dans la réussite d'un projet industriel.

II Outils et méthodes pour la conception des circuits analogiques et mixtes

II.1 Environnement logiciel de CAO

Les logiciels de CAO, apparus à la fin des années 1960, permettent de vérifier le fonctionnement d'un circuit électronique sans l'avoir jamais réalisé matériellement. Dans un environnement CAO, on dispose de divers outils pour passer du concept à la description

physique des circuits. Ces outils regroupent les interfaces de saisie de schéma, les simulateurs, les interfaces graphiques, les bibliothèques de modèles, les outils de layout et de vérification, etc.

Une interface permet généralement de saisir le schéma électrique en faisant appel aux schémas des composants élémentaires (transistors, diodes, capacités, etc. pour les circuits analogiques ; portes logiques, bascules, registres, etc. pour les circuits numériques). Ces composants existent dans des bibliothèques qui contiennent aussi des descriptions mathématiques ou modèles conformes aux technologies de fabrication. Un outil de vérification permet alors d'assurer la bonne connexion des composants et génère un fichier au format ASCII constitué d'une part d'une liste de connexions entre les composants et d'autre part du modèle propre à chaque composant. On obtient alors un système d'équations différentielles et algébriques (DAE). Les simulateurs résolvent ce système d'équations pour déterminer les potentiels en chaque nœud et les intensités de courant dans chaque branche du circuit.

Les interfaces graphiques permettent de visualiser les résultats obtenus ; elles nous offrent également la possibilité de les manipuler pour les représenter sous différentes formes.

Les concepteurs se servent des outils de layout pour générer une description physique des circuits. Des outils de vérification permettent de s'assurer que le layout obtenu est conforme à sa description schématique et qu'il respecte les règles de dessin imposées par le fondeur. Souvent, à ce stade, il est possible d'extraire du layout les capacités parasites introduites par les interconnexions. Ces capacités sont alors adjointes à la description schématique initiale et permettent une ultime simulation. Enfin, le layout est converti au format GDSII puis transmis au fondeur.

II.2 Evolution des simulateurs

II.2.1 Simulateurs numériques

Les premiers simulateurs numériques sont apparus dans les années 1970. Leur principe est de déterminer le fonctionnement booléen des circuits numériques constitués de portes logiques. Ce sont des simulateurs événementiels c'est-à-dire qu'ils ne se 'réveillent' que lorsqu'un événement se produit. Ceci leur permet d'être plus rapide que les simulateurs

analogiques. Ils analysent les circuits uniquement dans le domaine temporel discret en manipulant uniquement deux variables : l'état logique et le délai qu'il subit.

Parmi les simulateurs logiques on peut citer *QuickSimII* de Mentor Graphics et *Verilog-XL* de Cadence. Les nouvelles générations supportent la norme VHDL (*Very High speed integrated circuits Hardware Description Language*), langage de description matérielle. Sont apparus alors les simulateurs *ModelSim* de Mentor Graphics et *NC-Sim* de Cadence, etc.

II.2.2 Simulateurs analogiques

Les simulateurs analogiques permettent d'analyser les circuits électroniques au niveau transistor. Il existe différents types d'analyses : l'analyse DC qui détermine le point de fonctionnement statique, l'analyse transitoire qui détermine l'évolution temporelle des tensions et courants, l'analyse AC qui est une analyse fréquentielle petit signaux ou l'analyse RF pour les circuits Radio Fréquence, etc. Lors d'une analyse transitoire, et contrairement aux simulateurs numériques, les simulateurs analogiques analysent le circuit à chaque instant et non pas à chaque événement, ce qui les rend moins rapides. Par contre, leur précision est nettement supérieure puisqu'ils utilisent des modèles non linéaires et complets de transistors.

Le plus connu des simulateurs est *SPICE* (*Simulation Program with Integrated Circuit Emphasis*), élaboré par une équipe de chercheurs à l'université de Berkeley au début des années 1970. En 1975 est apparu *SPICE2* et à la fin des années 1980, Berkeley a développé la nouvelle version *SPICE3* qui est devenu le standard du marché. Les deux dérivés de *SPICE* qui ont connu le plus de succès sont *PSPICE* de MicroSim (apparu en 1984) et *HSPICE* de Avant!.

Les compagnies d'outils CAO ont développé chacune leurs propres simulateurs qui restent similaires à SPICE dans leurs fonctions et leurs applications, bien qu'ils reposent sur des algorithmes de résolution plus sophistiqués. On peut citer *Eldo* de Mentor Graphics, *Spectre* de Cadence, *SmartSpice* de Silvaco ou *Saber* de la compagnie Avant!.

II.2.3 Simulateurs mixtes

Avec un besoin de plus en plus important de concevoir des circuits mixtes, diverses générations de simulateurs mixtes se sont succédées [OUD00].

Une génération est apparue il y a environ cinq ans. Elle utilise la co-simulation entre les simulateurs analogiques et numériques à travers un *'back plane'* ou synchronisateur qui gère l'interface entre les parties analogiques et numériques. Ces deux parties sont simulées séparément et parallèlement par leur simulateur correspondant. Cette solution n'était pas très fiable et lourde à utiliser.

La dernière génération est basée sur un cœur de simulation unique. On intègre tous les algorithmes nécessaires pour simuler les parties analogiques et numériques directement dans le même code de programme ; on n'a donc plus besoin de co-simulation. Cette solution a été suscitée par l'apparition du standard VHDL-AMS, langage de description qui permet de concevoir l'analogique et le numérique dans le même environnement. *Saber* de Avant! fut un précurseur pour ce type de simulation et depuis deux ans, Mentor Graphics propose l'environnement de simulation mixte ADVanceMS.

II.3 Niveaux de description pour la simulation analogique

Il existe plusieurs niveaux pour décrire un circuit analogique. Le premier niveau est un niveau élémentaire dans lequel le circuit est décrit par des transistors, des résistances, des capacités, etc. Il est par conséquent appelé **niveau transistor**. C'est le niveau de description le plus bas ; il utilise des bibliothèques de modèles élémentaires tenant compte des technologies de fabrication.

Il existe un niveau de description plus élevé : c'est le **niveau comportemental**. Il s'agit de décrire le comportement du circuit électronique en traduisant l'évolution des signaux de sortie en fonction de ceux d'entrée sans tenir compte de sa structure interne. Cette description se fait à travers des fonctions et des lois mathématiques. Les lois de description sont ajustables au moyen de **paramètres génériques**. La vérification de la vraisemblance de ces lois se fait par comparaison avec des signaux mesurés sur un circuit réel ou bien simulés au niveau de description inférieure, souvent le niveau transistor.

On définit également un niveau de description plus élevé qui est le **niveau fonctionnel**. Il s'agit d'une description plus abstraite que le niveau comportemental dans laquelle on décrit le fonctionnement idéal du circuit avec des équations simplifiées, sans décrire par exemple les caractéristiques d'entrée et de sortie du circuit.

La description d'un circuit analogique sous forme comportementale ou fonctionnelle se fait en utilisant les langages de description matérielle (Hardware Design Language ou **HDL**).

II.4 Historique des langages de description matérielle

Le développement des langages de description matérielle a été fait dans un premier temps pour les circuits numériques. La nécessité de normaliser ces langages a donné naissance en 1987 à la norme IEEE 1076 définissant le VHDL (*Very High speed integrated circuits Hardware Description Language*). Une extension en 1999 de cette norme a donné naissance à la norme IEEE 1076.1 ou VHDL-AMS (*Very High speed integrated circuits Hardware Description Language for Analog and Mixed Systems*), qui inclut la norme VHDL et qui permet de modéliser aussi des circuits analogiques et mixtes.

Parallèlement à la norme VHDL s'est développée, avec quelques années de retard, la norme Verilog devant conduire prochainement au Verilog-AMS.

Parce que ces deux standards se sont partagés le champ des utilisateurs, il est vraisemblable qu'ils coexistent encore longtemps.

Le tableau 1. 1 regroupe quelques exemples de langages de description matérielle et de simulateurs mixtes de quelques compagnies fournissant des outils CAO.

tableau 1. 1 : principaux langages et simulateurs mixtes

Compagnie	Langage	Simulateur
Mentor	VHDL, Verilog HDL-A, VHDL-AMS	ADVance MS
Cadence	Verilog, VHDL SpectreHDL, Verilog-A, Verilog-AMS	AMS Designer
Avant !	Verilog, MAST	Star-MS
Dolphin Integration	VHDL-AMS	SMASH

II.5 Méthode traditionnelle de conception analogique

La description traditionnelle des circuits et systèmes analogiques s'appuie sur des composants de base qui sont les primitives de SPICE (transistors, diodes, capacités, etc.). La conception se base donc sur une description au premier niveau qui est le niveau transistor. Selon une méthode de conception traditionnelle, les transistors sont d'abord assemblés pour créer une fonction, laquelle est utilisée dans un bloc regroupant plusieurs fonctions et ainsi de suite jusqu'au système complet. Il s'agit d'une **méthode ascendante** ou **Bottom-Up**.

Dans les outils CAO, on dispose de bibliothèques de modèles de base qui sont attachés à la technologie de fabrication. L'évolution de ces technologies nécessite donc de mettre à jour les modèles disponibles.

Cependant, l'évolution des circuits intégrés fait que les simulateurs analogiques doivent simuler des circuits de plus en plus gros et avec des niveaux de précision de plus en plus fins avec un temps de simulation raisonnable. Ces nouvelles exigences font que les simulateurs traditionnels sont limités en performances. Si le nombre de transistors est trop grand, la simulation nécessite un temps très important. On peut imaginer alors le temps qu'il faut pour ajuster les paramètres d'un tel système afin de vérifier que son comportement est conforme aux spécifications initiales. Le passage à un niveau de description plus élevé par l'intermédiaire de la modélisation comportementale devient alors incontournable.

Pour montrer l'apport de la modélisation comportementale, prenons l'exemple d'une boucle à verrouillage de phase ou PLL (Phase Locked Loop). Il s'agit de la PLL 560B dont le schéma transistor va être présenté dans le chapitre 3. Elle est constituée d'une cellule Gilbert qui joue le rôle d'un comparateur de phase et d'un oscillateur à relaxation contrôlé en tension. Nous avons développé un modèle comportemental de la PLL en utilisant le langage de description matériel VerilogA. Le schéma transistor et le modèle ont été simulés dans l'environnement CADENCE. Le tableau 1. 2 résume les différents paramètres de simulation.

tableau 1. 2 : paramètres de simulation

Type de simulation	transitoire
Durée de la simulation	100 périodes
Simulateur	SpectreS
Type de machine	Ultra 5

Nous comparons dans le tableau 1. 3 les temps de simulations niveaux transistor et comportemental. Nous obtenons un gain de 12.2 en simulation comportementale. Ce gain peut être plus important pour des systèmes plus complexes.

tableau 1. 3 : Simulation d'une PLL : comparaison des temps de simulations

	Temps de simulation
Niveau transistor	65.2 s
Niveau comportemental	5.36 s
Gain en temps de simulation	12.2

II.6 Evolution des méthodes de conception

Compte tenu des limitations précédemment décrites, les méthodes de conception doivent évoluer pour intégrer les nouveaux besoins. Les objectifs peuvent se résumer en les points suivants :

- Résoudre le problème de temps de simulation important pour les systèmes électroniques qui sont de plus en plus complexes
- Détecter rapidement les erreurs de fonctionnement avant même d'établir une description niveau transistor
- Accélérer le cycle de conception des systèmes en réutilisant des blocs déjà conçus.

Pour aborder la conception d'un système complexe, mieux vaut alors procéder selon **une méthode descendante** ou **Top-Down** partant d'une description fonctionnelle du système et décomposant progressivement son architecture jusqu'au niveau transistor.

La figure 1 détaille les différentes étapes suivies dans une approche Top-Down : après une spécification du système à concevoir, on vérifie sa fonctionnalité avec une description fonctionnelle, donc avec 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 au niveau élémentaire (portes logiques ou niveau transistor). A partir de cette description, des outils de routage automatiques ou non permettent de générer le layout.

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. Ainsi, une modification de la technologie ne remet pas en cause les premières étapes de la conception.

Avec l'apparition des langages de description matérielle, la **méthode ascendante** ou **Bottom-Up** ne se limite plus au premier niveau élémentaire. Des modèles comportementaux des blocs constituant le système peuvent être extraits de leur description schématique. On pourra encore remonter dans les niveaux d'abstraction pour passer de la vérification fonctionnelle des blocs à celle de tout le système, comme illustré par la figure 1. 1.

En réalité, les concepteurs utilisent un mélange des deux approches Top-Down et Bottom-Up. En effet, le flot de conception Top-Down est loin d'être parfait et la passation d'une étape à une autre n'est pas toujours automatique. Nous pouvons aussi imaginer une conception Top-Down qui utilise des modèles de base issus de l'approche Bottom-Up.

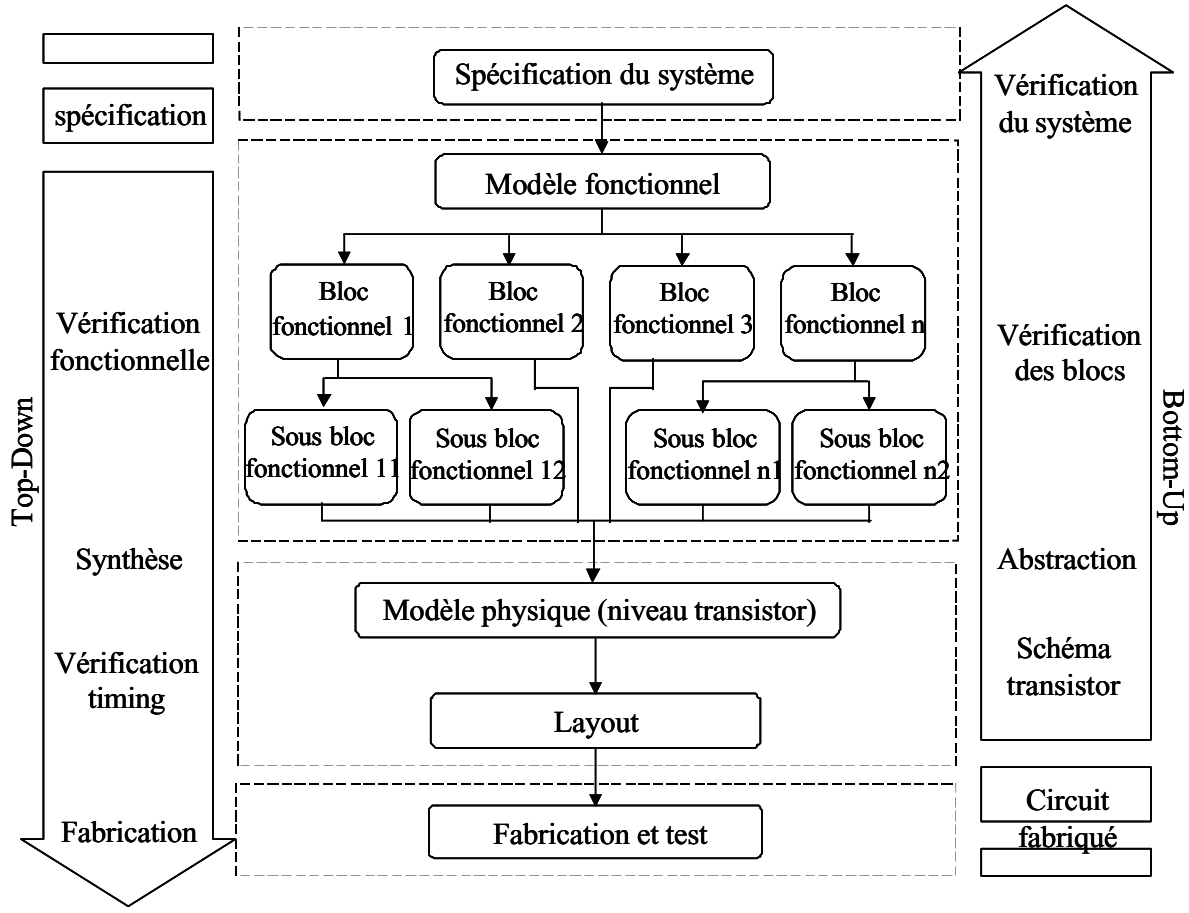

figure 1. 1 : Méthodologie de conception hiérarchique Top-Down et Bottom-Up

Dans les flux de conception Top-Down ou Bottom-Up, on peut mélanger dans une même description plusieurs niveaux d'abstraction. Ainsi, on peut simuler un système avec des blocs décrits au niveau fonctionnel et d'autres décrit au niveau transistor. C'est la technique de **multi abstraction**.

Pour accélérer le cycle de conception, il est par ailleurs souhaitable de re-utiliser (**Re-Use**) des blocs déjà conçus, éventuellement par un autre concepteur ou un autre organisme, ce qui remet en avant le délicat problème de la propriété intellectuelle (**IP**) de ces blocs. La modélisation comportementale prend dans ce cas tout son sens car le modèle peut devenir la « carte d'identité » d'un circuit sans que l'on connaisse son architecture. Une telle approche s'intègre bien dans les flux de conception hiérarchique Top-Down et Bottom-Up.

Ce nouveau flux de conception a été appliqué pour les circuits numériques dont la conception est devenue largement automatisée depuis environ une dizaine d'années. Cette automatisation est devenue possible grâce à l'apparition de la norme VHDL et des outils de synthèse automatique qui permettent de générer le schéma portes logiques à partir de la description de leurs fonctions. L'intégration de la méthode hiérarchique Top-Down dans le flux de conception des circuits constitue certainement un pas vers la synthèse analogique.

La compagnie Neolinear a développé un outil permettant de déterminer le dimensionnement des transistors qui est une opération importante dans la conception analogique. Trois autres compagnies, Barcelona Design, Analog Design Automation et Antrim Design ont proposé eux aussi sur le marché des outils de synthèse analogique. Cependant, aucun de ces produits n'adopte l'approche numérique et on est encore loin des performances des synthétiseurs numériques. Gary Smith de Dataquest, Stanford, dit qu'il n'a pas une grande confiance en ces produits puisque les méthodes de conception hiérarchique qui peuvent les supporter ne sont pas encore mises en place. Ces produits nécessitent donc beaucoup d'améliorations de point de vu performance mais aussi de point de vu facilité d'utilisation. La piste la plus prometteuse d'après Mickael Jackson, président du groupe R&D chez Avant! est celle qui permettra le passage de la phase Bottom-up à celle du Top-Down.

III Conclusion

Les nouvelles tendances de l'électronique sont l'intégration des System on Chip (SoC) et des ASIC mixtes, comprenant sur une surface toujours plus réduite des fonctions de plus en

plus complexes. Les outils CAO analogiques doivent alors évoluer pour rattraper leur retard sur la conception numérique, qui est de nos jours largement automatisée. Ceci permettra de réduire le temps de conception des ASIC mixtes et donc de diminuer leur coût.

La conception des circuits analogiques se fait encore souvent au niveau transistor, utilisant des simulateurs de type SPICE et selon une méthode de conception ascendante (Bottom-Up). Cependant, la complexité croissante des circuits augmente les temps de simulation et la simulation d'un système complet décrit au niveau élémentaire (transistor) est souvent impossible. Le flux de conception analogique doit donc évoluer pour adopter la méthode de conception hiérarchique descendante (Top-Down) qui part d'une description fonctionnelle du système en utilisant les langages de description matérielle pour descendre jusqu'à une description niveau transistor.

L'extension de la norme VHDL pour les circuits analogiques et mixtes (norme VHDL-AMS) a donné la possibilité de développer des modèles de haut niveau pour les circuits analogiques et mixtes et ceci au sein du même environnement. Mais le seul accès à ce langage standard ne répond pas aux besoins de la conception analogique et mixte. A présent, des simulateurs adaptés doivent être élaborés ou mûris, des bibliothèques de modèles restent à construire et de nouvelles méthodologies doivent être définies. En d'autre terme, l'objectif est de donner aux concepteurs les moyens méthodologiques et techniques leur permettant d'appliquer effectivement la méthode Top-Down pour les circuits analogiques et mixtes.

Pour contribuer à ce développement, nous nous sommes intéressés dans un premier temps à la formulation d'une méthodologie de modélisation en vue de la transmission du savoir-faire. Cette méthodologie doit aider les concepteurs à développer leurs propres modèles. Ceci aidera au développement des bibliothèques de modèles de circuits analogiques et mixtes, autre point indispensable à la mise en place de la conception Top-Down.

Nous avons alors développé une bibliothèque mixte pour les circuits RF (Radio-Fréquence). Elle contient les éléments de base qui permettent de modéliser des PLLs et des synthétiseurs de fréquence. Au cours de ce travail, un point précis a attiré notre attention.

En effet, un des critères important lors de la conception des circuits RF est l'optimisation du bruit de phase. Nous avons alors développé une approche de modélisation

pour tenir compte du bruit de phase dans nos modèles comportementaux et en particulier dans les oscillateurs.

Chapitre 2

Méthodes de modélisation comportementale

I Introduction

Avec le développement de la norme VHDL-AMS, il est devenu possible de décrire les systèmes analogiques et mixtes avec un langage de description matérielle standard. La question qui se pose alors est la suivante : comment modéliser de tels systèmes pour obtenir des modèles fiables répondant à un certain nombre de critères? Il est devenu par conséquent important et indispensable de développer une ou plusieurs méthodes de modélisation. Ces méthodes offriront aux concepteurs une démarche systématique leur permettant de développer leurs propres modèles en un délai raisonnable.

En reprenant la méthode de conception hiérarchique Top-Down et Bottom-Up, 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.

Nous allons commencer ce chapitre par donner les caractéristiques d'un modèle comportemental. Nous présentons ensuite les langages de modélisation VerilogA et VHDL-AMS en détaillant la structure d'un modèle.

Nous développons enfin les deux approches de modélisation schématique et fonctionnelle en illustrant à chaque fois par un exemple de modélisation.

II 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 transistor ;
- Une bonne précision par rapport à la réponse du circuit réel ;

- Pas de problèmes de divergence pour les différentes conditions d'opérations et les différents modes de simulations ;
- Avoir des paramètres génériques permettant d'adapter le modèle pour toute une classe de circuits similaires ;
- Avoir un gain de temps en simulation comportementale suffisamment important.

Les langages de modélisation définissent les structures des modèles comportementaux. Nous allons parler dans la suite des deux standards VHDL-AMS (norme IEEE 1076.1) et VerilogA (la norme VerilogAMS est en cours de standardisation).

III Langages de modélisation et structures des modèles

III.1 Verilog-A

Le langage de description matériel Verilog-A est un langage de haut niveau qui permet de décrire le comportement de plusieurs types de systèmes analogiques : électrique, mécanique, dynamique fluide ou thermodynamique [CAD97]. Cette description ce fait à travers des **modules**.

La structure d'un module est constitué de deux parties : une interface de déclaration et une description comportementale. Dans la première partie, on définit les entrées-sorties du module (mot clé **ports**) et leurs natures. On définit également des paramètres (mot clé **generic**). Cette partie constitue la vue externe du module ; les ports peuvent être connectés avec les ports d'autres modules et les paramètres peuvent être ajustés pour adapter le comportement du modèle développé avec une application spécifiée.

Dans la partie description comportementale, on exprime la caractéristique de transfert qui définit les grandeurs de sortie en fonction des entrées. La figure 2. 1 montre un exemple de modèle d'une diode écrit en Verilog-A. On peut distinguer les deux parties constituant le modèle.

```
            // Interface de déclaration du module //
module diode (np, nn);
    inout np, nn;
    electrical np, nn;
    parameter real area = 1;
    parameter real is = 1e-14;
    parameter real n = 2;
    parameter real cjo = 0;
    parameter real m = 0.5;
    parameter real phi = 0.7;
    parameter real tt = 1p;
            // Interface de description du module //
    real vd, id, qd;     // variables internes
analog begin
      vd = V(np, nn);
      id = area*is*(exp(vd/(n*$vt)) – 1);
      qd = tt*id+area*vd*cjo/pow((1-vd/phi),m);
      I(np,nn) <+ id + ddt(qd);
end
endmodule
```

figure 2. 1 : exemple de modèle en Verilog-A

Le langage Verilog-A définit plusieurs types et objets qui sont les entiers, les réels, les natures, les disciplines, les nœuds et les branches. Il dispose également de :

- Plusieurs opérateurs d'affectation, de calcul et de comparaison
- Une bibliothèque de fonctions mathématiques : Ln (x), log(x), exp(x), min(x), abs(x), pow(x, y), sin(x), tan(x), acos(x), tanh(x), atanh(x), etc.
- Des instructions permettant l'alternative, le choix multiple ou la répétition : if else, case, loop, repeat, while, for, etc.
- Des instructions pour détecter des événements : initial step, final step, cross, timer, etc.
- Des fonctions propres tel que :
  - Définition du pas de calcul
  - Détermination du temps de simulation
  - Fonction de bruit blanc
  - Fonction de génération d'un réel aléatoire
  - Transformée de Laplace

- Fonction d'affichage
- etc.

On peut également définir dans un module Verilog-A d'autres fonctions, écrire des procédures ou utiliser des fichiers.

La figure 2. 2 montre l'environnement CADENCE dans lequel on peut faire des simulations comportementales en utilisant le langage de description VerilogA.

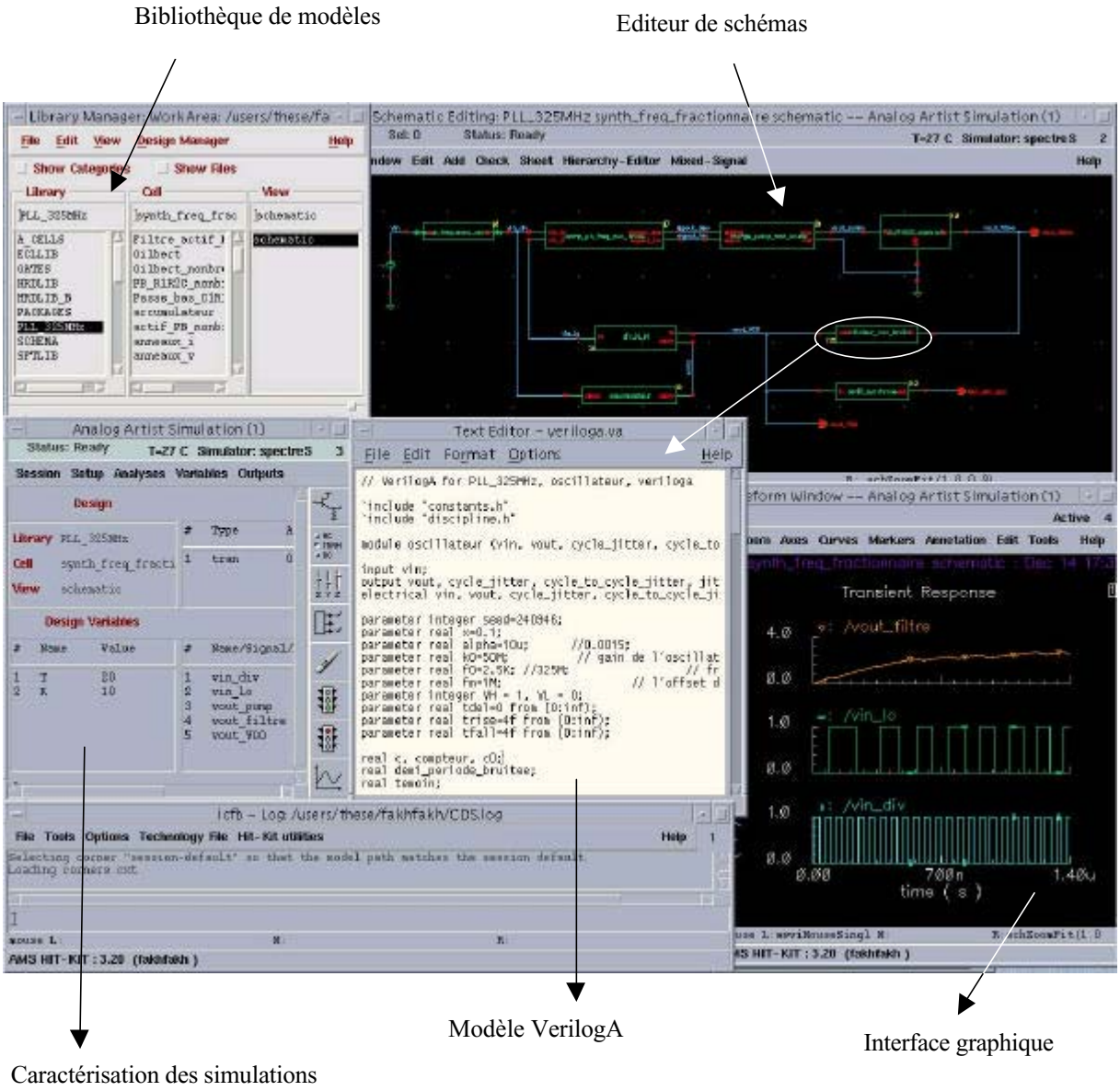

figure 2. 2: environnement de simulation CADENCE

Dans cet environnement, le concepteur dispose de plusieurs interfaces : une bibliothèque de circuits, un éditeur de schéma, une interface graphique pour représenter les courbes de simulations, etc. La bibliothèque contient les différents circuits à simuler. Ils peuvent être représentés sous différentes formes : schémas, symbole ou modèle VerilogA. Le schéma de la figure 2. 2 comprend plusieurs sous-blocs. Dans le cas d'une simulation comportementale, chaque sous-bloc est décrit par un modèle VerilogA. On a également la possibilité de décrire certains sous-blocs par des schémas transistors et effectuer une simulation mixte : comportementale et niveau transistor.

III.2 VHDL-AMS

Le langage de description matériel VHDL-AMS est l'extension du langage VHDL pour les circuits analogiques et mixtes. Il a donc été développé tout en gardant la norme VHDL. De nouvelles sémantiques de simulation ont été ajoutées pour supporter le comportement des circuits analogiques [ADV00].

Un modèle VHDL-AMS est constitué de deux parties principales : la spécification d'entité (mot clé **entity**) qui correspond à la vue externe du modèle et l'architecture de l'entité (mot clé **architecture**) qui est la vue interne [VHD99].

L'entité permet de définir les paramètres génériques (**generic**) et les entrées-sorties du modèle (les **ports)**, à travers lesquels il communique avec son environnement. Les différentes informations pouvant être échangées sont supportées par des ports de type **signal** pour une information à événement discret, **quantity** pour des informations analogiques orientées et **terminal** pour les informations analogiques de type nœuds de connexion.

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**. Dans un modèle VHDL-AMS, 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.

Avant la spécification de l'entité, on fait appel aux bibliothèques 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. Un exemple de modèle écrit en VHDL-AMS pour une capacité est montré à la figure 2. 3.

```
-- bibliothèques utilisées --

library disciplines, ieee;
use disciplines.electric_systems.ALL;
use IEEE.math_real.ALL;
-- spécification de l'entité --
entity capacity is
    generic (cap : real);
    port (terminal n1, n2 : electrical);
end entity capacity;
-- spécification de l'architecture --
architecture archi of capacity is
    quantity vc across n1 to n2;
    quantity ic through n1 to n2;
begin
    ic == cap*vc'dot;
end architecture archi;
```

figure 2. 3 : exemple de modèle VHDL-AMS pour une capacité

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 événements 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 de simulation qui doivent être synchronisés.

Le VHDL-AMS possède des types prédéfinis (*integer, real, bit, std_ulogic, bit_vector, boolean, severity_level, character, string, time*) et permet de définir d'autres types ou sous-types. Il définit également :

- Six classes d'objet qui permettent le transfert d'information : *constant, variable, signal, terminal, quantity et file.*

- Six classes d'opérateurs avec des niveaux de priorité : *logiques, relationnels, addition, signe, multiplication, et divers (**, abs, not).*

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

Les instructions simultanées servent à traiter l'information en temps continu. Elles sont évaluées à chaque point de simulation temporelle (ASP). Il existe l'instruction simultanée simple (==), l'instruction simultanée conditionnelle (*if use, elsif*), l'instruction simultanée sélective (*case, use*), l'instruction *null* et les procédures (*procedural*).

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'événement courant. Les processus sont des instructions concurrentes, l'affectation des signaux (<=), le *break* ou l'*assertion*.

Les instructions séquentielles sont évaluées en séquence dans le corps des processus. Les instructions principales sont l'affectation des variables (:=), l'affectation des signaux (<=), le *wait*, l'exécution conditionnelle (*if then, case*) et l'exécution itérative (*while, for, loop*).

La figure 2. 4 donne un exemple de modèle en VHDL-AMS qui illustre toutes ces différentes instructions :

```
-- bibliothèques utilisés --
library disciplines;
use disciplines.electric_systems.ALL;
use IEEE.math_real.ALL;

-- spécification de l'entité --
entity exemple is
    generic (ts : real := 1.0e-3);
    port (terminal out : electrical);
end entity exemple;

-- spécification de l'architecture --
architecture archi of exemple is
    signal t : real := 0.0;
    signal t0 : real := 0.0;
    signal Jn : real;
    quantity vout : across iout through out;
begin
    t <= now ;                              -- instructions concurrentes
    count <= t – t0 ;
P1 : process                                -- processus (instruction concurrente)
    variable seed1 : positive := 19823;
    variable seed2 : positive := 124;
    variable x : real;
begin
    wait until count'above(half_ts_noised) = true;    -- instruction séquentielle
                -- l'instruction wait déclenche le processus P1 --
    UNIFORM(seed1, seed2, x);
    Jn <= 2.0 * (x - 0.5);
    t0 <= t ;
end process P1;
    vout == 0.5 * ts * (1.0 + A*Jn);                  -- instruction simultanée

end architecture ;
```

figure 2. 4 : *exemple illustratif des différentes instructions en VHDL-AMS*

L'environnement de simulation ADVanceMS de Mentor Graphics est présenté par la figure 2. 5. Dans cet environnement, on peut visualiser le modèle VHDL-AMS, sa structure, les listes des processus, des objets et des paramètres. Une interface graphique permet de visualiser les courbes de simulations.

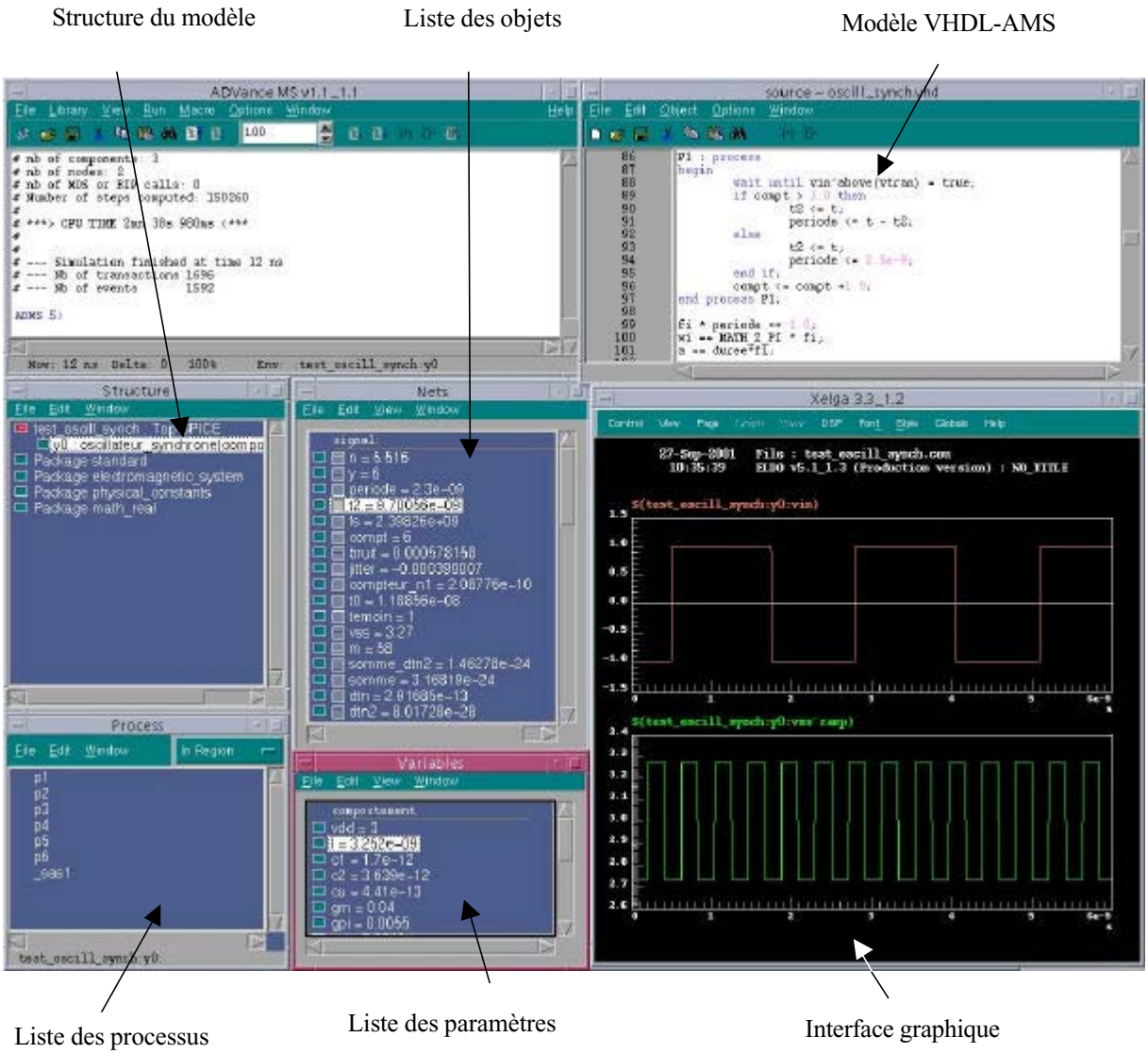

figure 2. 5 : environnement de simulation ADVanceMS

IV Modélisation comportementale

IV.1 Approche schématique

L'approche schématique consiste à développer des modèles comportementaux à partir des schémas niveau transistor des circuits à modéliser. Plusieurs démarches ont été présentées dans la littérature. On peut les classer selon deux stratégies : soit l'exploitation du schéma transistor, soit l'exploitation des résultats de simulation niveau transistor.

IV.1.1 Exploitation du schéma transistor

a) Simplification de circuits

Une première technique consiste à simplifier le schéma transistor. 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. Cette simplification se base sur l'analyse et la compréhension du schéma transistor.

Alan Mantooth propose une approche algorithmique pour la simplification des circuits [MAl93]. Elle consiste à classer les nœuds du circuit en nœuds linéaires, non linéaires et topologiques. L'analyse du cheminement du signal d'entrée vers la sortie permet de négliger les nœuds supposés non influents sur le comportement du circuit. On aboutit alors à un sous-ensemble de nœuds et par conséquent à un schéma simplifié.

Une fois le schéma simplifié, on applique 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.

Carsten Borchers propose un algorithme de simplification basé sur quatre étapes [BHE96]. La première consiste à résoudre les équations simples et indépendantes du système d'équations. La deuxième étape consiste à simplifier les termes de dérivées pour certaines variables qui n'introduisent pas une erreur importante en régime AC. Les variables sont ensuite remplacées par leurs valeurs moyennes calculées en analyse DC. Finalement, les équations différentielles sont transformées sous la forme de sommes de produits de variables.

Cette technique repose sur le développement d'algorithmes de simplification performants et adaptés à tous les circuits analogiques et mixtes.

IV.1.2 Exploitation des résultats de simulation

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

Jerzy Dabrowski propose par exemple une approche de linéarisation par morceaux [DPu98]. Michael Goedecke propose d'appliquer les méthodes d'estimation souvent utilisées pour identifier le comportement des systèmes automatiques [GHH95].

IV.1.3 Exemple de modélisation schématique

Pour illustrer l'approche schématique, nous avons développé un modèle comportemental pour la boucle à verrouillage de phase 560B [GrM93] en utilisant la technique de simplification du schéma transistor. Les modèles développés pour les circuits constituant la PLL sont découpés en trois blocs : un premier bloc définissant les caractéristiques d'entrée (courants et tensions d'entrée), un deuxième bloc contenant les caractéristiques de transfert et un dernier bloc définissant les caractéristiques de sortie (courants et tensions de sortie) [Mil97].

a) Modélisation de la PLL 560B

Une boucle à verrouillage de phase classique ou Phase Looked Loop (PLL) comprend un comparateur de phase, un filtre passe-bas et un oscillateur contrôlé en tension associés suivant le schéma de la figure 2. 6 :

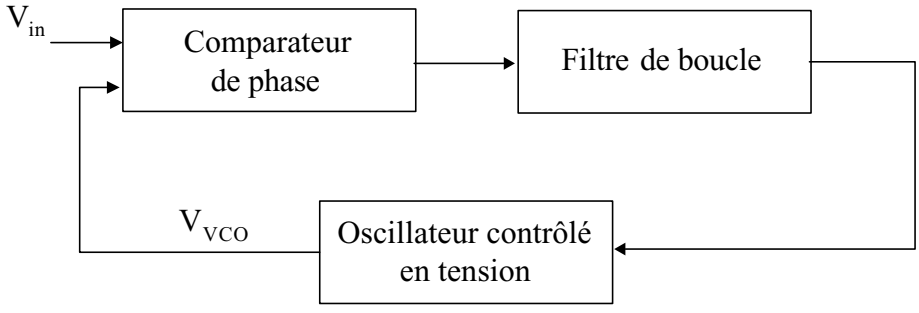

figure 2. 6 : schéma de principe d'une PLL

L'objectif d'un tel dispositif est de synchroniser le signal d'entrée (V_{in}) d'un oscillateur modulable en fréquence avec un signal de référence issu de l'oscillateur contrôlé

en tension (V_{VCO}) ; le synchronisme étant assuré par un asservissement de la phase des deux signaux. Une PLL est caractérisée par sa plage de capture, sa plage de verrouillage (dite aussi de maintien), son temps d'établissement, entre autres.

La plage de capture correspond à une plage de fréquence à l'intérieur de laquelle les signaux d'entrée et de référence se synchronisent. Une fois le dispositif synchronisé, la plage de maintien correspond à la plage de fréquence à l'intérieur de laquelle on peut faire varier, de façon infiniment lente, la fréquence du signal d'entrée sans que la PLL soit désynchronisée.

La PLL 560B est constituée d'une cellule Gilbert qui joue le rôle du comparateur de phase et d'un oscillateur à relaxation contrôlé en tension. Pour introduire un filtre passe bas, on ajoute dans le schéma de la cellule Gilbert une capacité comme illustré par le schéma de la figure 2.7.

a.i Comparateur de phase

La figure 2. 7 donne le schéma niveau transistor du comparateur de phase. Il s'agit d'un schéma simplifié dans lequel la source de courant a été remplacée par une source idéale I_0 et la source de tension par une tension constante V_0. Le schéma comprend deux parties, la cellule Gilbert (*a*) et un étage de sortie adaptateur de niveau et d'impédance (*b*).

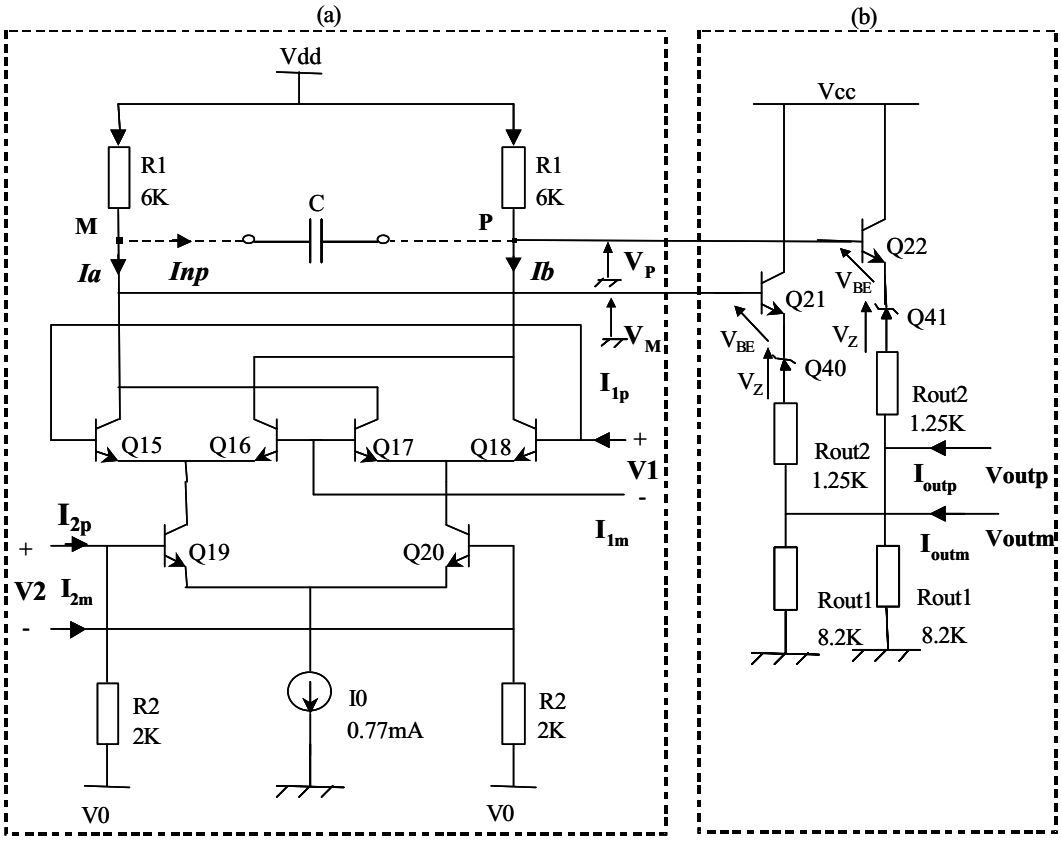

figure 2. 7 : schéma de la cellule Gilbert

Caractéristiques d'entrée

Les caractéristiques d'entrée définissent les courants d'entrée I_{1p}, I_{1m}, I_{2p} et I_{2m} en fonction des tensions d'entrée V_1 et V_2. Elles sont déterminées en appliquant le modèle simple des transistors et les lois de Kirchhoff. Elles sont données par les expressions suivantes :

$$I_{2p} = \frac{I_0 / \beta}{\left(1 + \exp\left(\frac{V_2}{V_T}\right)\right)} + \frac{V_2 - V_0}{R_2} \qquad (2.\ 1)$$

$$I_{2m} = \frac{I_0 / \beta}{\left(1 + \exp\left(\frac{-V_2}{V_T}\right)\right)} + \frac{V_2 - V_0}{R_2} \qquad (2.\ 2)$$

$$I_{1p} = \frac{I_0/\beta}{\left(1+\exp\left(\frac{-V_2}{V_T}\right)\right)*\left(1+\exp\left(\frac{V_1}{V_T}\right)\right)} + \frac{I_0/\beta}{\left(1+\exp\left(\frac{V_2}{V_T}\right)\right)*\left(1+\exp\left(\frac{V_1}{V_T}\right)\right)} \quad (2.\ 3)$$

$$I_{1m} = \frac{I_0/\beta}{\left(1+\exp\left(\frac{-V_2}{V_T}\right)\right)*\left(1+\exp\left(\frac{-V_1}{V_T}\right)\right)} + \frac{I_0/\beta}{\left(1+\exp\left(\frac{V_2}{V_T}\right)\right)*\left(1+\exp\left(\frac{-V_1}{V_T}\right)\right)} \quad (2.\ 4)$$

β étant le gain en courant des transistors supposés identiques.

Caractéristiques de transfert

Calcul des courants I_a et I_b

En utilisant le modèle d'Ebers et Moll pour les transistors, on exprime les courants de collecteur des différents transistors du schéma de la cellule Gilbert par les relations suivantes :

$$Ic20 = \frac{I_0}{1+\exp\left(\frac{V_2}{V_T}\right)} \quad \text{et} \quad Ic19 = \frac{I_0}{1+\exp\left(\frac{-V_2}{V_T}\right)}$$

$$Ic15 = \frac{Ic19}{1+\exp\left(\frac{-V_1}{V_T}\right)} = \frac{I_0}{\left[1+\exp\left(\frac{-V_1}{V_T}\right)\right]\left[1+\exp\left(\frac{-V_2}{V_T}\right)\right]}$$

$$Ic16 = \frac{Ic19}{1+\exp\left(\frac{V_1}{V_T}\right)} = \frac{I_0}{\left[1+\exp\left(\frac{V_1}{V_T}\right)\right]\left[1+\exp\left(\frac{-V_2}{V_T}\right)\right]}$$

$$Ic17 = \frac{Ic20}{1+\exp\left(\frac{V_1}{V_T}\right)} = \frac{I_0}{\left[1+\exp\left(\frac{V_1}{V_T}\right)\right]\left[1+\exp\left(\frac{V_2}{V_T}\right)\right]}$$

$$Ic18 = \frac{Ic20}{1+\exp\left(\frac{-V_1}{V_T}\right)} = \frac{I_0}{\left[1+\exp\left(\frac{-V_1}{V_T}\right)\right]\left[1+\exp\left(\frac{V_2}{V_T}\right)\right]}$$

En négligeant les courants de bases des transistors Q21 et Q22, on obtient :

$$\begin{aligned} I_a &= I_{c15} + I_{c17} \\ I_b &= I_{c16} + I_{c18} \end{aligned} \quad (2.\ 5)$$

D'où l'expression donnant la différence entre les courants I_a et I_b :

$$I_a - I_b = I_{11} * \left[\tanh\left(\frac{V_1}{2 V_T} \right) \right] \left[\tanh\left(\frac{V_2}{2 V_T} \right) \right] \quad (2.\ 6)$$

Calcul des tensions de sorties

En appliquant les lois de Kirchhoff aux nœuds P et M, on peut exprimer leurs tensions correspondantes V_P et V_N par les expressions suivantes :

$$V_P = V_{dd} - R_1 \left(I_b + I_{np} \right)$$

$$V_M = V_{dd} - R_1 \left(I_a - I_{np} \right)$$

Les tensions de sortie à vide V_{outp0} et V_{outm0} sont respectivement exprimées en fonction de V_P et V_M par les expressions suivantes :

$$V_{outp0} = \frac{R_{out1}}{R_{out1} + R_{out2}} \left(V_P - V_{BE} - V_Z \right) \quad (2.\ 7)$$

$$V_{outm0} = \frac{R_{out1}}{R_{out1} + R_{out2}} \left(V_N - V_{BE} - V_Z \right) \quad (2.\ 8)$$

On supposera V_{BE} constante.

Caractéristiques de sortie

Les caractéristiques de sortie définissent les tensions de sortie respectivement en fonction des courants de sortie I_{outp} et I_{outm} et les tensions à vide V_{outp0} et V_{outm0} exprimées précédemment. Elles sont données par l'expression suivante :

$$V_{outp} = V_{outp0} + R_{out}\, I_{outp} \quad (2.\ 9)$$

$$V_{outm} = V_{outm0} + R_{out}\, I_{outm} \quad (2.\ 10)$$

$$avec \cdot R_{out} = R_{out1} // \left(R_{out2} + \frac{1}{gm} \right)$$

g_m étant la transconductance des transistors Q21 et Q22 supposés identiques. Sa valeur est déterminée à partir des simulations niveau transistor.

a.ii Oscillateur contrôlé en tension (VCO)

Le schéma niveau transistor de l'oscillateur contrôlé en tension est donné par la figure 2. 8. Les sources de courant sont aussi remplacées par des sources idéales. Le schéma est composé de deux parties : un convertisseur tension courant (a) et un oscillateur à relaxation (b).

Pour modéliser le VCO, on applique comme précédemment la décomposition du modèle en trois blocs : caractéristiques d'entrée, caractéristiques de transfert et caractéristiques de sortie.

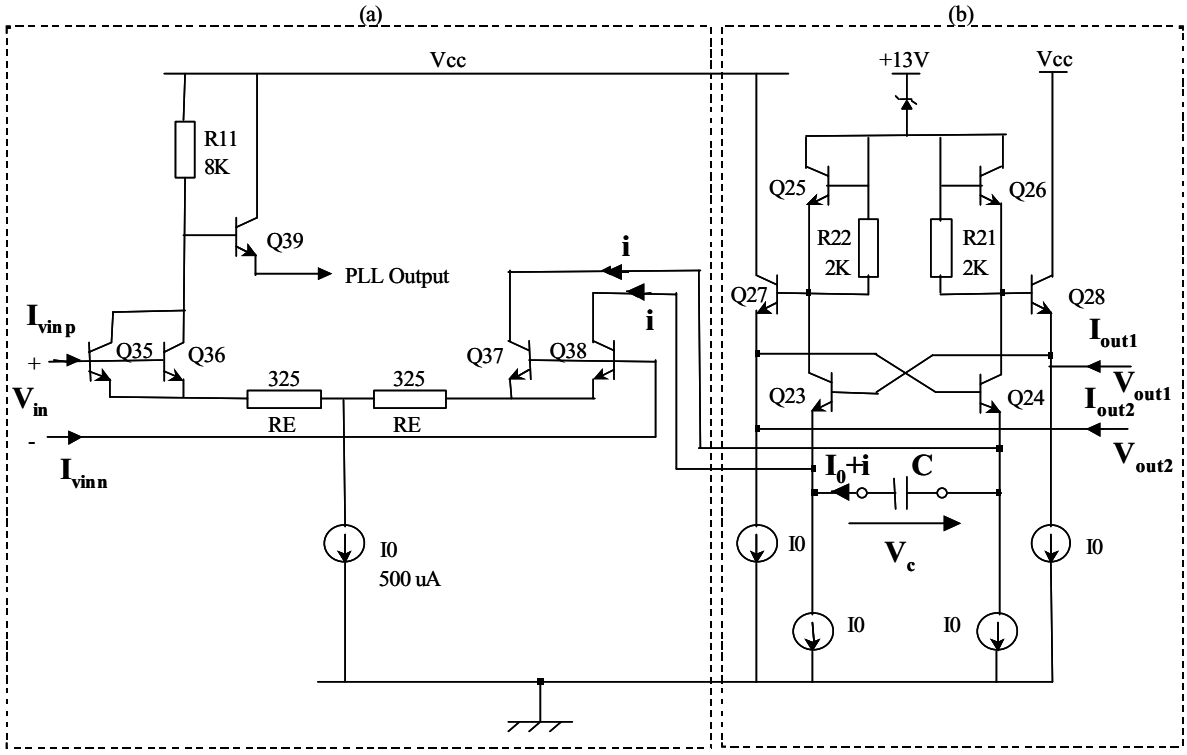

figure 2. 8 : schéma niveau transistor du VCO

Caractéristiques d'entrée

Les courants d'entrée I_{vinp} et I_{vinn} sont définis en fonction de la tension d'entrée V_{in} par les deux expressions approchées suivantes [Dev94] :

$$I_{vinp} = \frac{I_0/\beta}{1+\exp\left(\frac{2\,V_{in}}{R_{10} * I_0}\right)} \qquad (2.\ 11)$$

$$I_{vinn} = \frac{I_0/\beta}{1+\exp\left(\frac{-2\,V_{in}}{R_{10} * I_0}\right)} \qquad (2.\ 12)$$

β est le gain en courant des transistors d'entrée supposés identiques.

Caractéristiques de transfert

Calcul du courant i

Pour modéliser la partie (a) du schéma transistor du VCO, on exprime le courant de collecteur 'i' des transistors Q37 et Q38 en fonction de la tension d'entrée V_{in}. En appliquant les modèles simples des transistors, on trouve la relation approchée suivante [Dev94] :

$$i = \frac{I_0}{1+\exp\left(\frac{2\,V_{in}}{R_{10}\,I_0}\right)} \qquad (2.\ 13)$$

Calcul de la fréquence de sortie

L'oscillateur de relaxation de la partie (b) délivre en sortie une tension dont la fréquence varie linéairement avec le courant 'i'. Son principe de fonctionnement repose sur la charge et la décharge de la capacité C par un courant constant égal à $I_0 + i$.

Les transistors (Q_{27}, Q_{24}) et (Q_{28}, Q_{23}) sont alternativement saturés ou bloqués, ce qui détermine le sens du courant traversant la capacité. Lorsque (Q_{27}, Q_{24}) sont saturés, la tension Vc se charge linéairement suivant la pente $\frac{I_0 + i}{C}$ jusqu'à ce qu'elle atteigne la tension base-émetteur V_{BE} des transistors supposés identiques ; alors les transistors (Q_{27}, Q_{24}) se bloquent et se sont (Q_{28}, Q_{23}) qui deviennent saturés. Le courant traversant C est alors inversé et la tension Vc se décharge jusqu'à $-V_{BE}$.

La fréquence du signal de sortie f_{VCO} s'exprime en fonction de i et C par l'expression suivante [GrM93] :

$$f_{VCO} = \frac{I_0 + i}{4\,C\,V_{BE}} \qquad (2.\ 14)$$

La partie du modèle développé en VHDL-AMS pour modéliser l'oscillateur à relaxation est présentée par la figure 2. 9 :

```
                                    -- initialisation --
if domain = QUIESCENT_DOMAIN
use    icc==Ic0;
                        -- calcul de la tension aux bornes de la capacité --
else   vc == icc'integ / c;
end use;
                                  -- Caractéristiques d'entrée --
ip == I0/beta1/(1.0+exp(-2.0*(vp-vm)/(RE*I0)));
im == I0/beta2/(1.0+exp(-2.0*(vm-vp)/(RE*I0)));
                                --  Caractéristiques de transfert --
vin == vp - vm ;
ic == I0 + ip * 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)                      -- test du sens de courant i --
use    outm0 == vddl;
       outp0 == 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 outm0 == vddh;
          outp0 == vddl;
       if (vc <= -vbe)
       use          icc == ic;
       else         icc == -ic;
       end use;
end use;
```

figure 2. 9 : modélisation de l'oscillateur à relaxation

Après une phase d'initialisation, le modèle calcule la tension V_c aux bornes de la capacité C en intégrant la pente $\frac{I_0 + i}{C}$. Le courant i change de signe et passe à –i lorsque la tension V_c atteint la valeur V_{BE}. Dans ce cas, on intègre $\frac{I_0 - i}{C}$ jusqu'à ce que V_c atteigne la valeur $-V_{BE}$. Le courant –i change alors de signe et repasse à i et le cycle recommence de nouveau.

La tension de sortie est un signal carré qui bascule de niveau à chaque inversion du sens de courant i.

Les modèles VHDL-AMS complets du comparateur de phase et du VCO sont donnés en annexe.

b) Simulation comportementale de la PLL 560B

b.i Simulation du comparateur de phase

Le modèle VHDL-AMS du comparateur de phase a été simulé avec ADVanceMS. La figure 2. 10 présente la réponse temporelle du modèle en appliquant à l'entrée deux tensions différentielles carrées déphasées de $\pi/2$. En ajoutant la capacité C, la tension de sortie est filtrée autour d'une valeur moyenne qui correspond au déphasage entre les tensions d'entrées comme le montre la première courbe de la figure 2. 11 (déphasage de $\pi/6$). Le temps de réponse dépend de la valeur de C, ici choisie égale à 4.42nF.

Nous comparons également sur la figure 2. 11 les réponses du modèle et du circuit transistor. La courbe d'erreur qui donne la différence entre les deux tensions de sortie montre un bon accord entre les deux types de simulation.

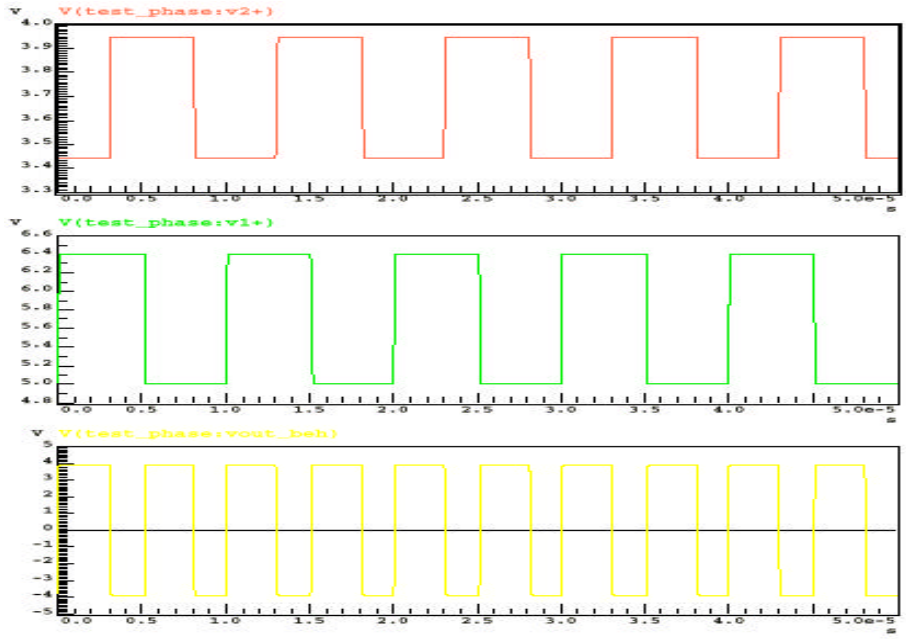

figure 2. 10 : réponse du comparateur de phase

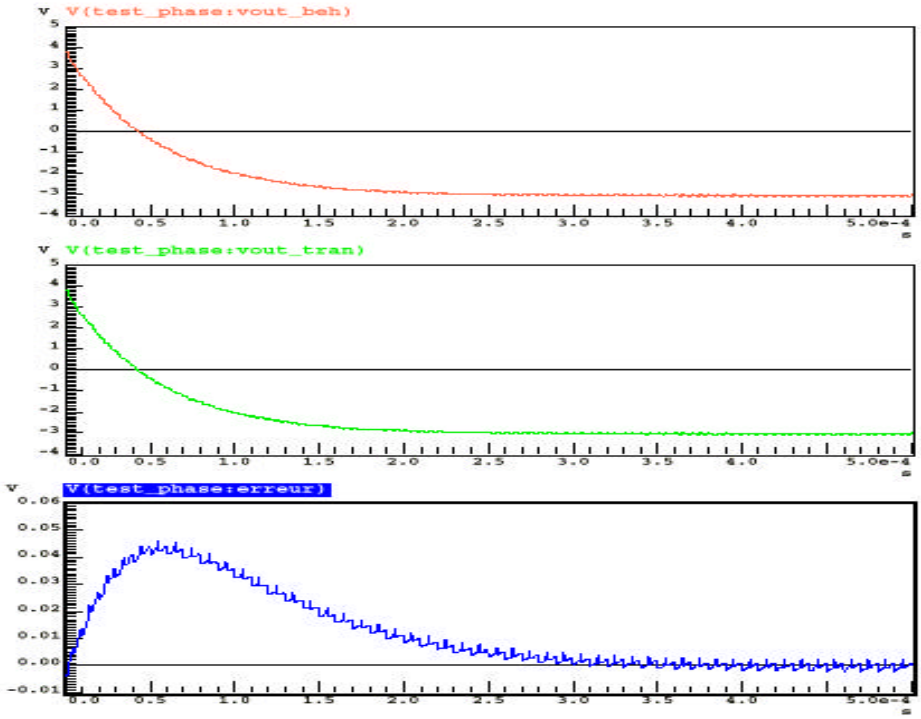

figure 2. 11 : filtrage de la tension de sortie

En faisant varier le déphasage entre les tensions d'entrée V_1 et V_2, nous traçons la caractéristique de transfert du comparateur de phase. Cette caractéristique est comparée à celle obtenue avec des simulations niveau transistor. L'erreur relative entre les deux courbes est faible, ce qui valide le modèle développé.

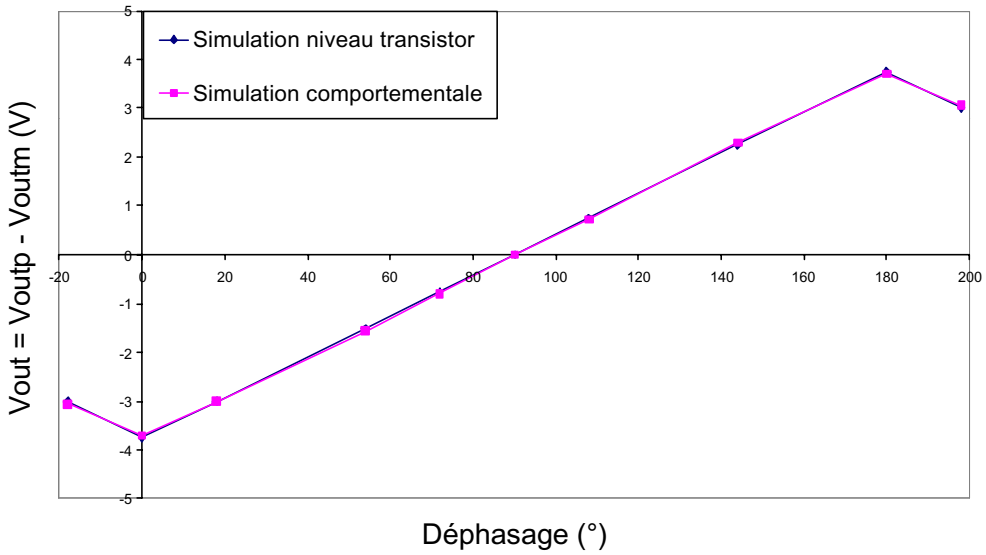

figure 2. 12 : caractéristique de transfert du comparateur de phase

Pour une durée d'analyse égale à 50 périodes d'entrées et pour un déphasage égal à π/6, le temps de la simulation comportementale est de 12.55 s. Celui de la simulation du schéma transistor est de 35.15 s. On obtient par conséquent un gain en simulation comportementale égale à 2.8.

b.ii Oscillateur contrôlé en tension

Le modèle VHDL-AMS du VCO a été aussi simulé avec ADVanceMS. Pour une fréquence propre du VCO fixée à 100 kHz, une tension d'entrée constante V_{in} est appliquée à l'entrée. On la fait varier de –0.6V à +0.6V. En mesurant pour chaque tension appliquée la fréquence de sortie, on trace la caractéristique de transfert du VCO :

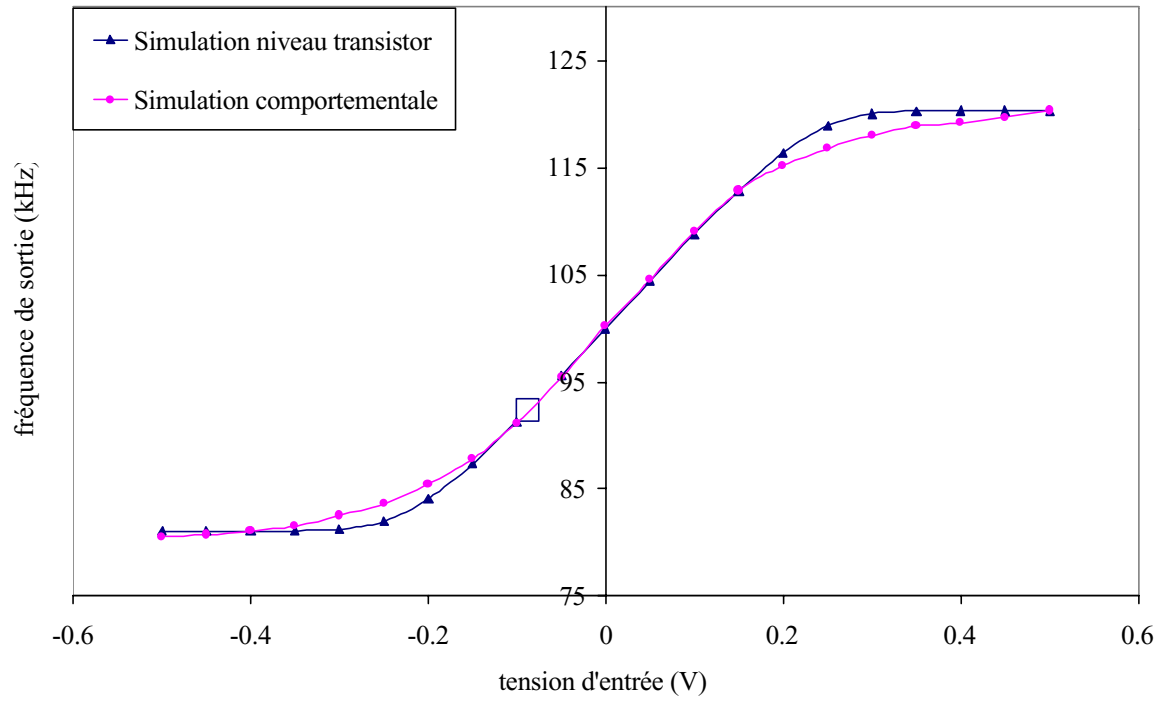

figure 2. 13 : caractéristique de transfert du VCO

Nous comparons dans la figure 2. 13 les performances du modèle et du circuit transistor. Il y a un bon accord dans la zone linéaire de la caractéristique de transfert ; ce qui valide le modèle développé.

Pour une durée d'analyse égale à 50 périodes de la tension de sortie, le temps de la simulation comportementale est de 3.04 s. Celui de la simulation du schéma transistor est de 15.19 s. Ceci donne un gain en simulation comportementale égal à 5.

b.iii Simulation de la PLL

Un modèle comportemental de la PLL est obtenu en regroupant le modèle du comparateur de phase et celui du VCO. Des simulations au niveau transistor et comportementales ont été faites pour retrouver la plage de capture de la PLL.

Pour déterminer la plage de capture, on fait varier la fréquence du signal appliqué à l'entrée de 70 kHz jusqu'à 136 kHz et on vérifie pour chaque fréquence si la PLL est accrochée ou pas. La figure 2. 14 compare les deux plages de capture trouvées. On remarque un bon accord entre les deux résultats trouvés.

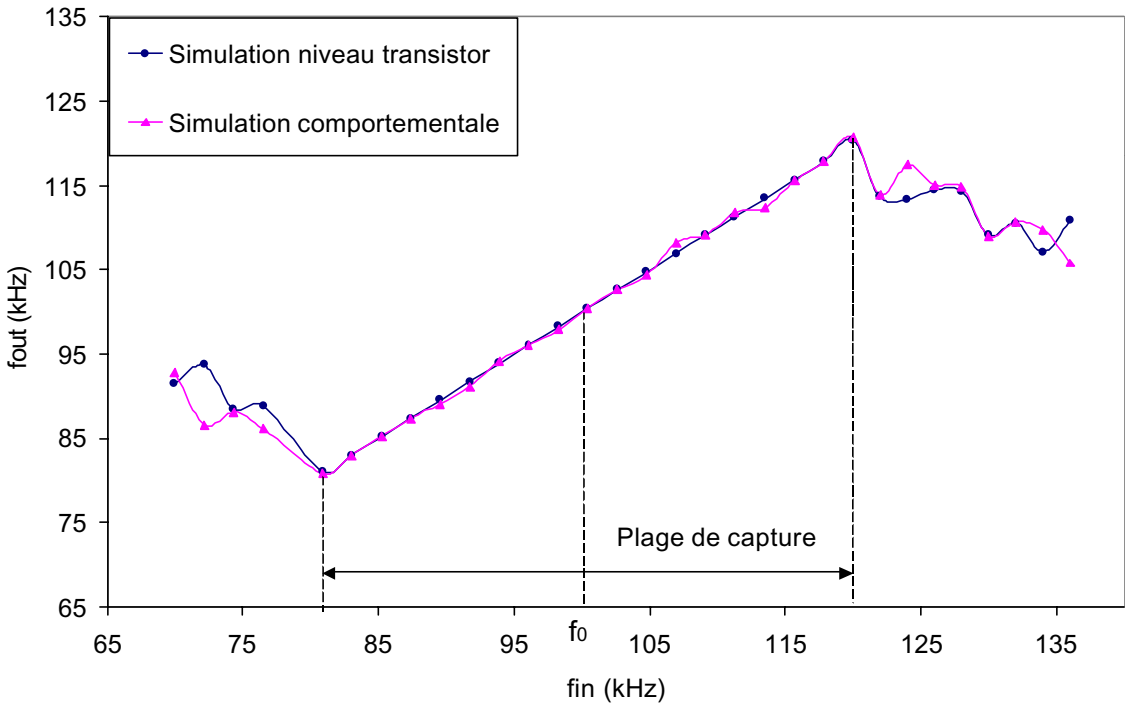

figure 2. 14 : plage de capture de la PLL 560B

Le *tableau 2. 1* compare les temps de simulations comportementale et niveau transistor de la PLL pour un temps réel égal à 1 ms correspondant à l'accrochage et l'établissement du régime permanent :

tableau 2. 1: comparaison des temps de simulation

Temps réel	1 ms
Simulation niveau transistor	5 mn 47.72 s
Simulation comportementale	2 mn 43.05 s
Gain en simulation comportementale	2.13

IV.2 Approche fonctionnelle

La deuxième approche utilisée pour la modélisation comportementale peut être qualifiée d'approche fonctionnelle. Contrairement à l'approche schématique, elle ne s'appuie pas sur l'analyse du schéma ni sur l'exploitation des résultats de simulations au niveau transistor. Elle consiste à analyser la fonction du circuit.

En effet, chaque circuit en électronique réalise une fonction plus ou moins complexe, dépendant de paramètres plus ou moins nombreux. Par exemple un comparateur compare les

tensions d'entrées ; un convertisseur tension/fréquence convertit une tension en une fréquence et un oscillateur délivre en sortie une tension périodique, etc.

Par expérience, on s'aperçoit que dans la structure de ce genre de modèles, des sous-fonctions apparaissent fréquemment. Nous les avons recensés et nous décomposons, dans le paragraphe suivant, la démarche de modélisation.

IV.2.1 Démarche systématique

Un modèle communique avec son environnement à partir des bornes d'entrées/sorties. Les paramètres génériques ajustables de l'extérieur permettent d'adapter le modèle à une même classe de circuits.

Dans notre démarche systématique, la structure fondamentale du modèle fonctionnel est décomposée en trois parties comme illustré par la figure 2. 15. Dans une première partie, les variables d'entrées sont détectées. Une deuxième partie sert à calculer les paramètres des signaux de sorties à partir des variables d'entrées et des paramètres génériques. Dans une dernière partie, on génère les signaux de sorties.

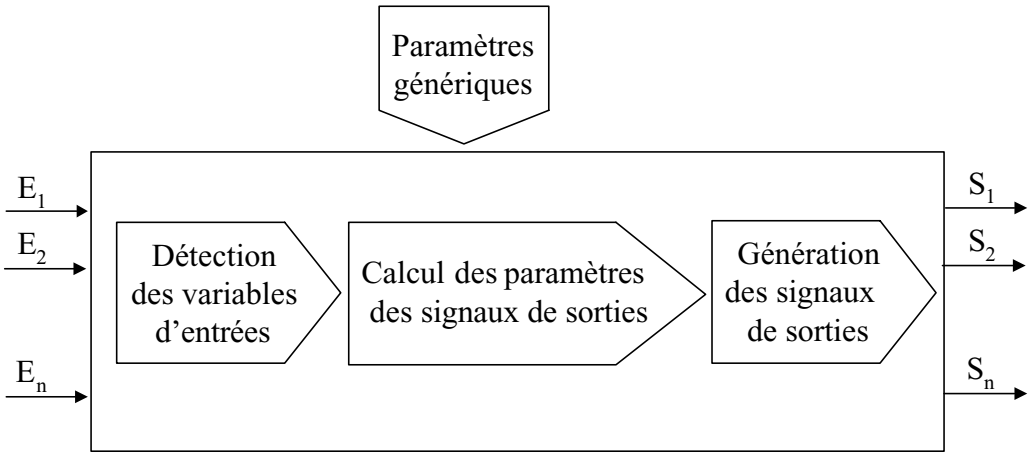

figure 2. 15 : structure fondamentale d'un modèle fonctionnel

a) Détection des variables d'entrées

Il s'agit de détecter les informations qu'apportent les signaux d'entrées et qui vont être utiles pour déterminer les paramètres des signaux de sorties. Nous avons recensé principalement cinq types de variables à détecter :

- tension d'entrée

- courant d'entrée
- front (montant et descendant) du signal d'entrée
- fréquence du signal d'entrée
- durée d'une impulsion d'entrée

a.i Détection des tensions et des courants d'entrée

VHDL-AMS comme VerilogA permettent de détecter une différence de potentiel ainsi qu'un flux entre deux nœuds.

Cas VHDL-AMS

On définit en VHDL-AMS des quantités (**quantity**) par l'instruction ci-dessous où *inp* et *inm* sont deux nœuds d'entrée définis comme '**terminal**' :

```
quantity vin across iin through inp to inm;
```

Dans cet exemple, on définit la tension *vin* et le courant *iin* entre les deux nœuds *inp* et *inm*.

Pour un signal de type numérique, on définit des signaux d'entrée de type bit :

```
signal vin : in bit;
```

Dans ce cas, *vin* prend les valeurs '0' ou '1'.

Cas VerilogA

En VerilogA, les deux nœuds d'entrée *inp* et *inm* sont définis comme '**electrical**'. L'instruction **V**(inp, inm) définit la tension entre ces deux nœuds et l'instruction **I**(inp, inm) le courant allant de *inp* vers *inm*.

a.ii Détection des fronts montants et descendants

Cas VHDL-AMS

Pour un signal de type analogique, la détection d'un front montant ou le passage de la tension d'entrée V_{in} par une tension *seuil* dans le sens positif peut être fait par l'instruction suivante :

```
wait until vin'above (seuil) = true;
```

Par contre, pour détecter un front descendant ou le passage de la tension d'entrée par une tension *seuil* dans le sens négatif, on peut utiliser l'instruction suivante :

```
wait until vin'above (seuil) = false;
```

Dans le cas d'un signal de type numérique, on utilise les deux instructions suivantes, respectivement pour la détection d'un front montant ou descendant :

```
wait until vin = '1';      ou      wait until vin = '0';
```

Ces instructions peuvent servir à déclencher un processus.

Cas VerilogA

En VerilogA, on utilise les instructions suivantes :

- Détection du front montant : @cross(V(inp, inm) - seuil, +1))
- Détection du front descendant : @cross(V(inp, inm) - seuil, -1))

a.iii Détection de fréquence

Les langages VHDL-AMS et VerilogA ne disposent pas d'instruction ou de fonction permettant de détecter les fréquences des signaux d'entrées. Nous avons alors écrit des modèles qui détectent la période d'entrée et qui en déduit la fréquence.

Le principe repose sur la détection de deux passages successifs de la tension d'entrée par une tension *seuil* et ceci dans le même sens comme l'illustre la figure 2. 16 :

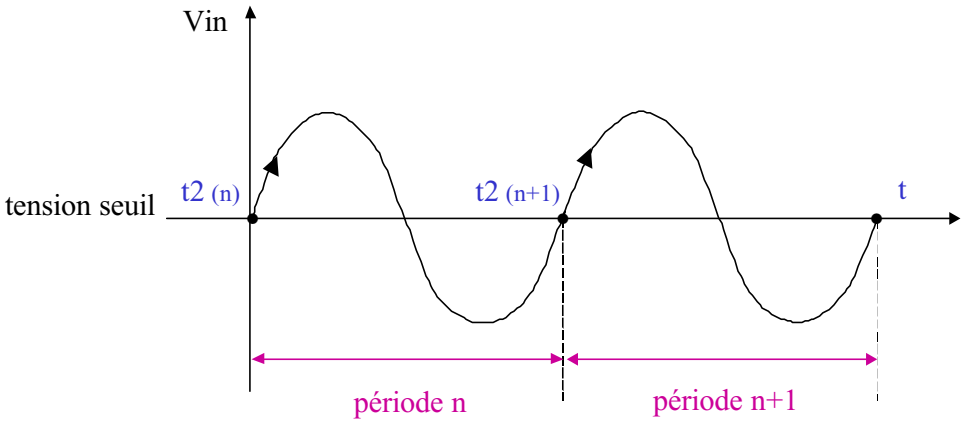

figure 2. 16 : principe de la détection de la période

Ce principe permet de calculer à chaque passage par la tension *seuil* la période d'entrée et donc de détecter chaque changement de fréquence.

Le processus développé en VHDL-AMS et qui est appelé *detection_frequence*, est présenté figure 2. 17. Il calcule la fréquence d'un signal d'entrée analogique.

```
      generic (seuil : real := 0.0;
               T0 : real := 1.0e-3);
      quantity vin across iin through inp to inm;
      quantity t : real;
      signal t2 : real := 0.0;
      signal periode : real;
      signal frequency : real;
          -- Calcul du temps écoulé de la simulation --
t == now ;
detection_frequence : process
      variable temoin : real := 0.0;
begin
      -- Détection d'un front montant du signal d'entrée --
      wait until vin'above(seuil) = true;
      if temoin > 1.0 then
            -- Calcul de la période du signal d'entrée --
               periode <= t – t2;
                 -- initialisation de la valeur de t2 --
               t2 <= t;
      else
               t2 <= t;
               periode <= T0;
      end if;
      temoin <= temoin + 1.0;
      frequency <= 1.0/periode;
end process detection_frequence;
```

figure 2. 17 : calcul de la fréquence d'entrée

La variable '*temoin*' est incrémentée à chaque fronts montants. Au premier front, le processus ne calcule pas la période d'entrée et considère la période initialisée par le modèle.

Si le signal d'entrée est numérique, le processus *detection_frequence* est déclenché sur les fronts du signal d'entrée *vin* comme suit :

```
detection_frequence : process (vin)
```

Dans le cas d'un modèle VerilogA, le principe de modélisation reste le même, en utilisant l'instruction déjà présentée pour détecter les fronts montants.

a.iv Détection d'une durée d'impulsion

L'information utile apportée par le signal d'entrée peut être la durée d'impulsion. Pour détecter cette caractéristique, nous avons écrit le processus *duree_impulsion*.

Le principe consiste à détecter deux passages successifs de la tension d'entrée par une tension seuil comme illustrée par la figure 2. 18.

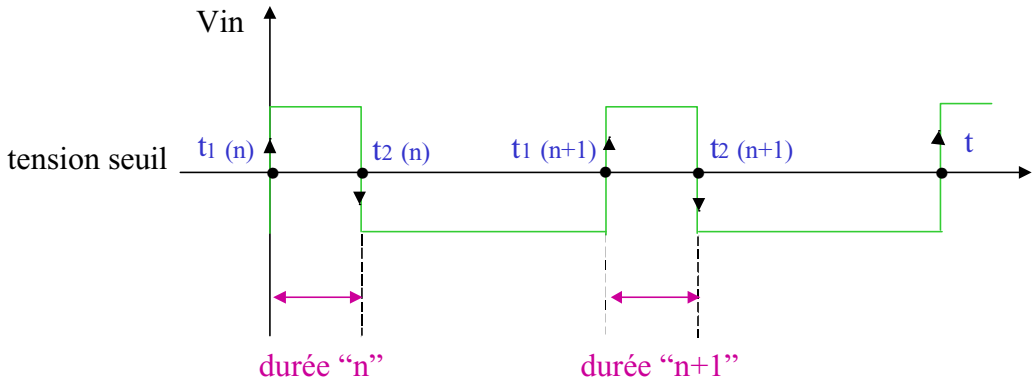

figure 2. 18 : principe de la détection de la durée d'une impulsion

Le processus développé est présenté figure 2. 19 :

```
        generic (seuil : real := 0.0);
        quantity vin across iin through inp to inm;
        quantity t : real;
        signal t1 : real := 0.0;
        signal duree_impulsion : real;
        t == now ;
duree_impulsion : process
begin
        -- Détection d'un front montant du signal d'entrée --
                -- => détermination du temps t1 --
        wait until vin'above(seuil) = true;
        t1 <= t;
      -- Détection d'un front descendant du signal d'entrée --
              -- => calcul de la durée d'impulsion --
        wait until vin'above(seuil) = false;
```

```
        duree_impulsion <= t – t1;
end process duree_impulsion;
```

figure 2. 19 : calcul de la durée d'impulsion

On note que le principe de modélisation reste le même si le signal d'entrée est numérique ou s'il s'agit d'un modèle VerilogA.

Les modèles de calcul de la fréquence d'entrée et de la durée d'impulsion sont rangés dans une bibliothèque appelée *bibliothèque de base*. Ils serviront au développement d'une bibliothèque de modèles fonctionnels.

b) Calcul des paramètres des signaux de sortie

La deuxième partie du modèle fonctionnel forme généralement la partie principale dans laquelle on calcule les paramètres des sorties en fonction des caractéristiques d'entrées et des paramètres génériques. Il s'agit de déterminer les tensions, les courants, les fréquences ou les formes des signaux de sorties.

Prenons l'exemple d'un diviseur de fréquence. Ce circuit divise la fréquence d'entrée par un rapport N fixe. La fréquence du signal de sortie est dans ce cas un paramètre de sortie à calculer à partir de la fréquence d'entrée. Si on veut avoir en sortie un signal carré de rapport cyclique variable, comme c'est le cas dans le modèle ci-dessous, la largeur des impulsions de sortie sont aussi un paramètre à calculer à partir de la fréquence d'entrée et du paramètre définissant le rapport cyclique.

La figure 2. 20 montre la partie du modèle du diviseur de fréquence N développé, dans laquelle est calculée la fréquence et la durée des impulsions de sortie. Le modèle intégrale sera présenté dans le chapitre suivant.

```
    generic (N, cyclic_ratio : real);
architecture behavior of frequency_divider is
    signal thigh : real := cyclic_ratio*initial_period;
    signal period_sortie : real := initial_period;
begin
calcul_periode_sortie : process
begin
    wait until Vin'above(Vthreshold) = true;
                        -- calcul de la période de sortie --
```

```
    periode_sortie <= N*periode_entree;
                        -- calcul de la durée d'une impulsion --
    thigh <= N*cyclic_ratio* periode_entree;
end process calcul_periode_sortie;
end architecture behavior;
```

figure 2. 20: calcul des paramètres du signal de sortie d'un diviseur de fréquence

c) Génération des signaux de sorties

La dernière partie du modèle fonctionnel sert à générer les signaux de sorties. Deux cas peuvent être envisagés :

- Soit le signal de sortie dépend directement du signal d'entrée, comme pour un amplificateur, un filtre ou un amplificateur. Dans ce cas la génération du signal de sortie est liée à la fonction même du circuit :

  Amplificateur : $V_s = A V_e$

  Filtre : $\tau \frac{dV_s}{dt} + V_s = A V_e$

  Multiplieur : $V_s = A V_s V_e$

  Il s'agit d'une génération commandée par le signal d'entrée.

- Soit le signal de sortie dépend de paramètres caractéristiques du signal d'entrée, mais pas directement du signal d'entrée lui-même. Entrent également dans cette catégorie les générateurs 'libres', n'ayant pas de bornes d'entrée. Dans ce cas, il s'agit le plus souvent de générer un signal périodique en sortie.

Nous avons introduit alors dans notre *bibliothèque de base* des blocs de génération de signaux tels que des générateurs de tension sinusoïdale ou carrée.

Nous exposons par le modèle de la figure 2. 21 un exemple de générateur de tension sinusoïdale. Les paramètres nécessaires pour générer un tel signal sont l'amplitude, la fréquence et éventuellement la condition initiale.

```
Library disciplines;
Use disciplines.electromagnetic_system.all;
Library IEEE;
Use IEEE.math_real.all;

entity gene_V_sin is
      generic (amplitude : real := 1.0; freq : real := 1.0e3;
                  dc : real := 0.0);
      port (terminal in : electrical);
end entity gene_V_sin;

architecture compotement of gene_V_sin is
      quantity v_sin across i_sin through in;
begin
      v_sin == amplitude*SIN(2.0*Math_pi*freq*NOW)+dc;
end architecture;
```

figure 2. 21 : exemple de générateur de tension sinusoï dale

IV.2.2 Exemple de modélisation fonctionnelle

Pour donner un exemple de modélisation fonctionnelle, nous reprenons le cas d'une PLL de structure analogue à la PLL 560B. Le modèle développé nous a permis de comparer les deux approches de modélisation.

a) Modélisation fonctionnelle

a.i Comparateur de phase

La valeur moyenne de la tension du signal de sortie d'un comparateur de phase est proportionnel à la différence de phase entre les deux entrées V_1 et V_2. Nous prendrons, par analogie avec la cellule de Gilbert utilisée dans la PLL 560B, un OU exclusif suivi d'un filtre passe-bas.

En se basant sur la table de vérité de la porte OU exclusif, nous avons développé le modèle fonctionnel suivant :

```
Library disciplines;
Use disciplines.electromagnetic_system.all;
Library IEEE;
Use IEEE.math_real.all;

entity OU_exclusif is
        generic (VH : real := 4.62; VL : real := -4.62);
        port (terminal E1, E2, out : electrical);
end entity OU_exclusif;

architecture fonctional of OU_exclusif is
        constant VT : real := (VH + VL) / 2.0;
                -- détection des variables d'entrée --
        quantity VE1 across IE1 through E1;
        quantity VE2 across IE2 through E1;
        quantity Vout across Iout through out;
begin
                -- génération de la tension de sortie --
        IE1 == 0.0;
        IE2 == 0.0;
        If ((VE1 < VT) and (VE2 < VT)) use Vout == VL;
        elsif ((VE1 > VT) and (VE2 > VT)) use Vout == VL;
        elsif ((VE1 > VT) and (VE2 < VT)) use Vout == VH;
        else Vout == VH;
        end use;
end architecture fonctional;
```

figure 2. 22 : modèle fonctionnel du comparateur de phase

Le filtre associé au comparateur de phase est un filtre passe bas du premier ordre. Sa transmittance $H_f(p)$ s'écrit :

$$H_f(p) = \frac{K_f}{1 + \tau p}$$

La fréquence de coupure du filtre f_c est définie par : $f_c = \frac{1}{2\pi\tau}$.

Le modèle fonctionnel développé est le suivant :

```
Library disciplines;
Use disciplines.electromagnetic_system.all;
Library IEEE;
Use IEEE.math_real.all;

entity passe_bas is
        generic (gain : real := 1.0; fc : real := 1.0e3);
        port (terminal inp, inm :  electrical;
   terminal outp, outm : electrical);
end entity passe_bas;
architecture fonctional of passe_bas is
                -- détection des variables d'entrée --
        quantity vin across iin through inp to inm;
        quantity vout across iout through outp to outm;
        quantity tau : real;
begin
        iin == 0.0;
        tau == 1.0/(2.0*MATH_PI*fc);
                -- génération de la tension de sortie --
        tau*vout'dot+vout == gain*vin;
end architecture fonctional;
```

figure 2. 23 : modèle fonctionnel du filtre passe bas

a.ii Oscillateur contrôlé en tension

Un oscillateur contrôlé en tension ou VCO délivre en sortie une tension périodique dont la fréquence f_s varie proportionnellement à son entrée V_{in} :

$$f_s(t) = f_0 + K_{VCO} V_{in}(t) \qquad (2.\ 15)$$

avec f_0 la fréquence centrale du VCO et K_{VCO} son gain.

Le modèle du VCO que nous avons développé et qui est présenté figure 2. 24 contient les trois blocs envisagés pour un modèle fonctionnel. Dans la partie détection des paramètres d'entrée, on détecte la tension d'entrée *Vin*. Dans la deuxième partie, on calcule la fréquence de sortie f_s à partir de l'équation (2. 15). La fréquence centrale f_0 du VCO et son gain K_{VCO} sont considérés comme paramètres génériques. Dans la dernière partie, on génère en sortie un signal carré de rapport cyclique égal à 0.5.

```
entity VCO is
        generic (Vhigh, Vlow, trise, tfall, vco_gain : real; vco_period : time);
        port (terminal Tinput, Toutput : electrical);
end entity VCO;

architecture behavior of VCO is
                        -- détection de la tension d'entrée --
        quantity Vin across Tinput to electrical_ground;
        quantity Vout across Iout through Toutput to electrical_ground;
        signal period : time := vco_period;
        signal semaphore_1 : real := 1.0;
begin
            -- limites de la zone linéaire de la caractéristique du VCO --
    if (Vin>=-0.21 and Vin<=0.21) use phi == Vin;
    elsif (Vin<-0.21) use phi == -0.21;
    else phi == 0.21;
    end use;
    end use;
    semaphore_1 <= -semaphore_1 after (0.5*period);
output_period : process
begin
        wait until semaphore_1'event;
                        -- calcul de la période de sortie --
        period <= real2time(1.0/((1.0/time2real(vco_period))+Vin*vco_gain));
end process output_period;
                        -- génération du signal de sortie --
        Vout == semaphore_1'ramp(trise, tfall);
end architecture behavior;
```

figure 2. 24 : modèle fonctionnel du VCO

b) Résultats de simulations

Nous avons appliqué les mêmes paramètres de simulation considérés pour le premier modèle de la PLL 560B. Les résultats de simulations sont comparés à ceux obtenus avec le schéma niveau transistor.

b.i Comparateur de phase

La figure 2. 25 représente la sortie du comparateur de phase en appliquant à l'entrée deux tensions V_1 et V_2 déphasées de $\pi/2$. La figure 2. 26 présente la sortie filtrée pour un

déphasage de -π/6. En comparant cette sortie à celle du schéma transistor, on obtient une erreur relative faible (courbe 3 de la figure 2. 26).

Les caractéristiques de transfert du comparateur de phase obtenues respectivement avec des simulations comportementales et niveau transistor sont comparées dans la figure 2. 27. L'erreur relative est faible entre les deux courbes, ce qui valide le modèle fonctionnel développé.

Pour une durée d'analyse égale à 50 périodes d'entrées et un déphasage égal à π/6, le temps de la simulation comportementale est de 6s. Celui de la simulation du schéma transistor est de 35.2s. Ceci nous donne un gain en simulation comportementale de l'ordre de 6.

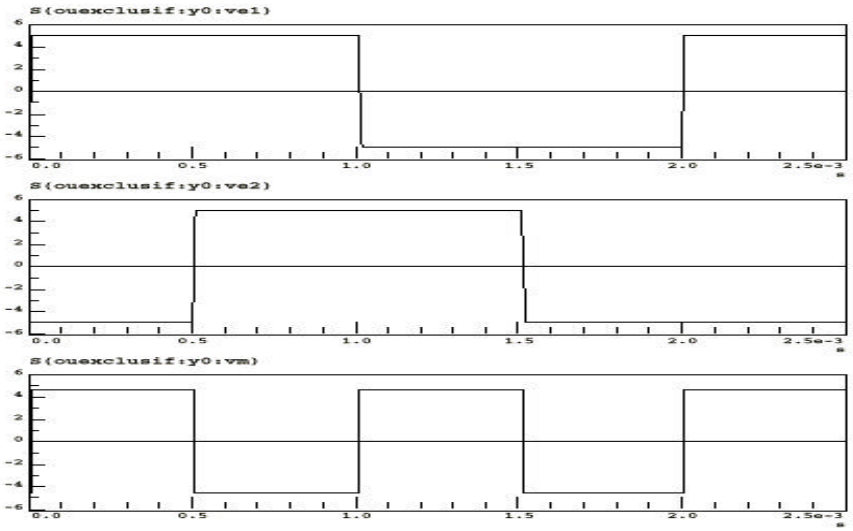

figure 2. 25 : réponse du comparateur de phase

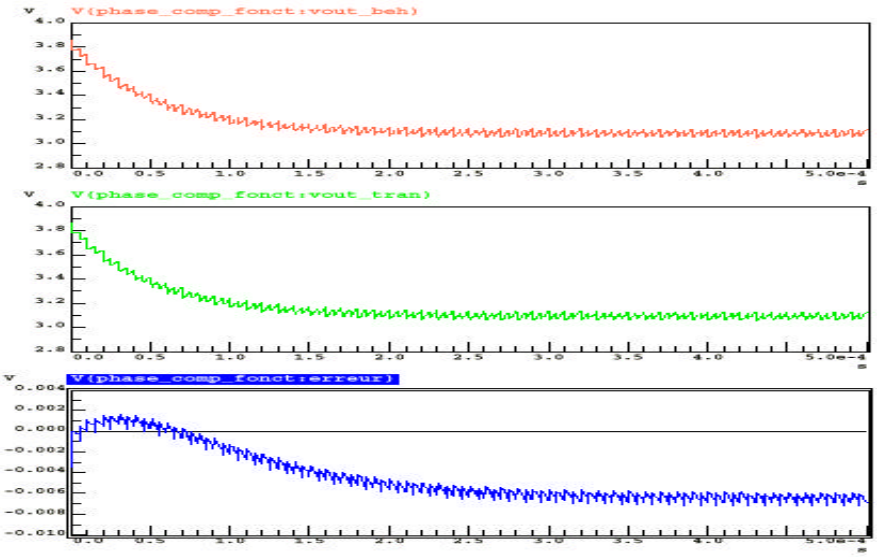

figure 2. 26 : filtrage de la tension de sortie

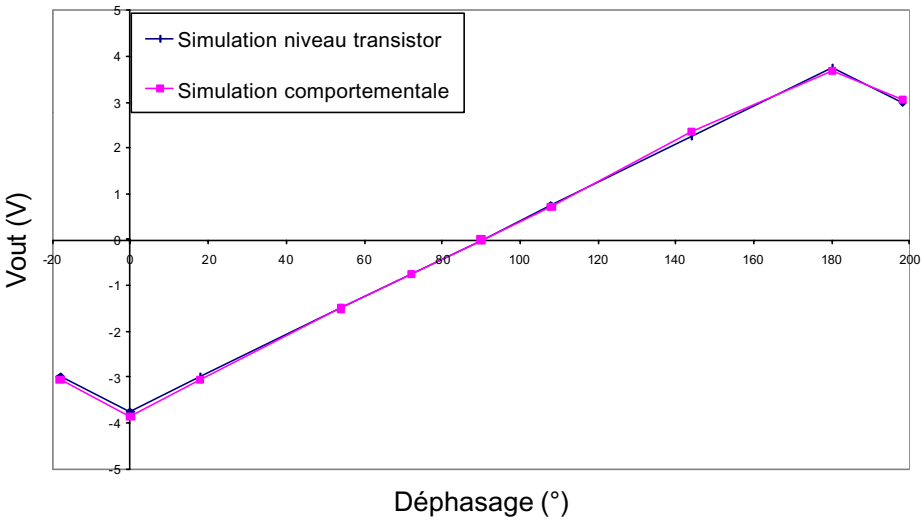

figure 2. 27 : caractéristique de transfert du comparateur de phase

b.ii Oscillateur contrôlé en tension

Nous traçons dans la figure 2. 28 la courbe donnant la caractéristique de transfert du VCO. Elle est comparée à celle obtenue avec des simulations niveau transistor. Il y a un bon accord entre les deux courbes dans la zone linéaire du VCO.

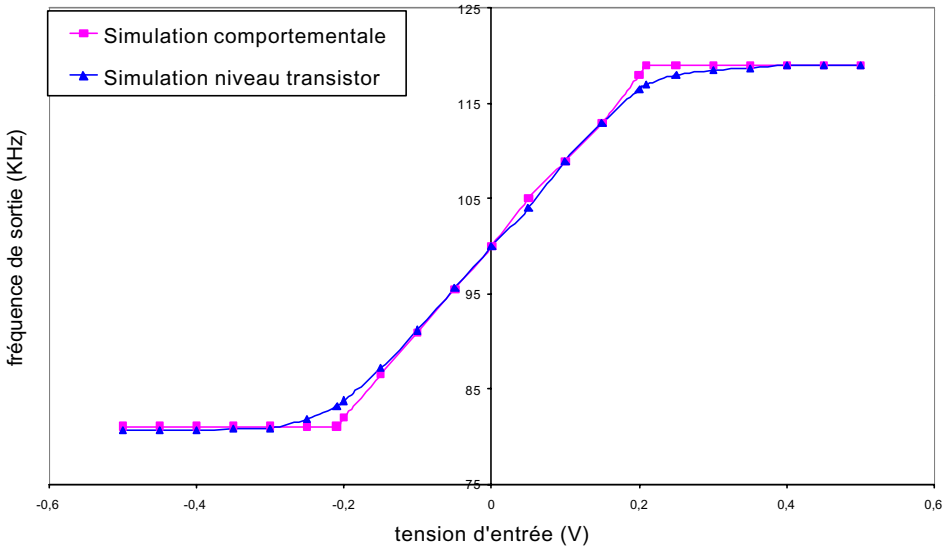

figure 2. 28 : caractéristique de transfert du VCO

Pour une durée d'analyse égale à 50 périodes de la tension de sortie, le temps de la simulation comportementale est de 0.68 s. Celui de la simulation du schéma transistor est de 15.19s. Ceci nous donne un gain en simulation comportementale de l'ordre de 22.

b.iii Simulation de la PLL

La simulation du modèle fonctionnel de la PLL nous a permis de déterminer la plage de capture. Elle est comparée à celle de la simulation niveau transistor. Il y a un bon accord entre les courbes, ce qui valide notre modèle.

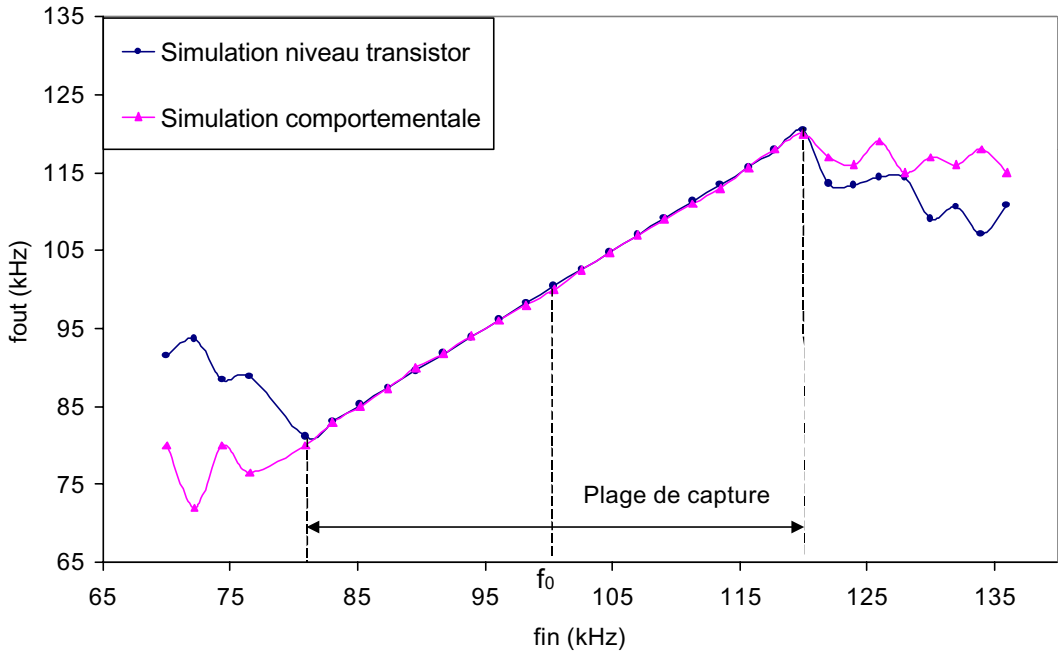

figure 2. 29: plage de capture de la PLL

Le *tableau 2. 2* compare les temps de simulations comportementale et niveau transistor de la PLL pour un temps réel égal à 1 ms correspondant à l'accrochage et l'établissement du régime permanent :

tableau 2. 2: comparaison des temps de simulation

Temps réel	1 ms
Simulation niveau transistor	5 mn 47.72 s
Simulation comportementale	1 mn 0.34 s
Gain en simulation comportementale	5.8

IV.3 Comparaison entre les approches schématique et fonctionnelle

Nous avons comparé les performances des deux approches schématique et fonctionnelle de point de vu rapidité, précision, utilisation de paramètres génériques et réutilisabilité pour des circuits similaires. Les résultats sont regroupés dans le tableau suivant :

tableau 4.1: comparaison des deux approches de modélisation

	Approche schématique	Approche fonctionnelle
Rapidité	☺☺	☺☺☺
Précision	☺☺☺	☺☺
Paramètres génériques	☺	☺☺☺
réutilisation	☺	☺☺☺

Un modèle développé selon l'approche schématique donne généralement des résultats précis par rapport aux résultats des simulations niveau transistor. Cependant, il est moins rapide par rapport aux modèles fonctionnels pour deux raisons. Premièrement parce qu'il utilise les modèles simples des transistors en appliquant les lois de Kirchhoff, ce qui aboutit à des équations différentielles relativement complexes ; et deuxièmement parce que sa taille augmente avec la taille du circuit. Ses paramètres sont généralement non génériques, compromettant sa réutilisation pour des circuits de même classe. L'approche schématique est plutôt adaptée à la phase Bottom-Up de la conception hiérarchique pour extraire des modèles raffinés.

Un modèle fonctionnel présente une meilleure performance en terme de rapidité du fait qu'il repose sur l'analyse de la fonction du circuit et non sur l'application des modèles simples des transistors. Il présente également une meilleure performance en terme de réutilisation du fait qu'il utilise des paramètres génériques. Pour ces raisons, nous l'avons adopté pour développer notre bibliothèque de modèles comportementaux pour les circuits radio-fréquences.

Chapitre 3

Bibliothèque comportementale pour la synthèse de fréquence

I Introduction

Un des freins principaux à l'évolution du flot de conception analogique et mixte est l'absence de bibliothèques de modèles standards et bien documentés. Ce chapitre expose le développement d'une bibliothèque comportementale pour les circuits radio fréquences (RF). Les modèles développés ont été validés en comparant leurs performances à celles des simulations au niveau transistor et des mesures faites sur des circuits fabriqués. Les modèles sont ici présentés en VHDL-AMS mais une version VerilogA existe également.

II Description de la bibliothèque de modèles

Le contenu de la bibliothèque est donné par le tableau 3. 1 dans lequel les modèles sont répartis selon plusieurs catégories.

tableau 3. 1 : contenu de la bibliothèque RF

Catégorie	Modèles	Niveau de description	Interface de description
Comparaison de phase	Comparateur phase-fréquence	Comportemental	Analogique ou numérique
	Pompe de charge	Comportemental	Analogique ou mixte
Filtrage	Filtre passe bas	Comportemental	Analogique
	Filtre passe haut	Comportemental	Analogique
	Filtre passe bande	Comportemental	Analogique
	Filtre coupe bande	Comportemental	Analogique
Division de fréquence	Diviseur N	Comportemental	Analogique ou numérique
	Diviseur N/N+M	Comportemental	Analogique ou numérique
	Accumulateur	Comportemental	Analogique ou numérique
	Diviseur fractionnaire	Structurel	Analogique ou numérique
Oscillateurs	Oscillateur contrôlé en tension	Comportemental	Analogique ou mixte
	Oscillateurs synchrones	Comportemental	Analogique ou numérique
Synthèse de fréquence	Boucles à verrouillage de phase	Structurel	Analogique ou mixte
	Synthétiseur de fréquence fractionnaire	Structurel	Analogique ou mixte

Les modèles structurels sont obtenus par association de plusieurs modèles comportementaux.

Les circuits de PLL ou des synthétiseurs de fréquence peuvent être constitués de blocs analogiques ou numériques. Seuls les filtres utilisés sont généralement des circuits analogiques. Nous avons alors développé pour chaque modèle VHDL-AMS de notre bibliothèque deux types de descriptions : un premier type dans lequel les interfaces des modèles sont analogiques. Dans ce cas, les entrées/sorties sont définies comme étant des terminaux électriques. Dans le deuxième type de description, les interfaces sont numériques et les entrées/sorties sont définies comme étant des signaux de type bit.

Durant la simulation, le modèle à interface analogique fait appel au simulateur analogique et le modèle à interface numérique fait appel au simulateur numérique. La simulation numérique est plus rapide mais elle est moins réaliste et plus abstraite puisque elle n'introduit pas des temps de montés et de descentes pour les signaux d'entrées/sorties. Nous ne présentons dans ce chapitre que les modèles à interface analogique, en sachant que le principe de modélisation reste exactement le même pour les modèles à interface numérique.

II.1 Comparaison de phase

La catégorie *comparaison de phase* contient un modèle de comparateur phase-fréquence et un modèle de pompe de charge. Ce principe de comparaison de phase est en effet le plus utilisé en synthèse de fréquence.

II.1.1 Comparateur phase fréquence

a) Principe de fonctionnement

Un exemple de comparateur phase-fréquence ou Phase Frequency Detector (PFD) est donnée par la figure 3. 1 :

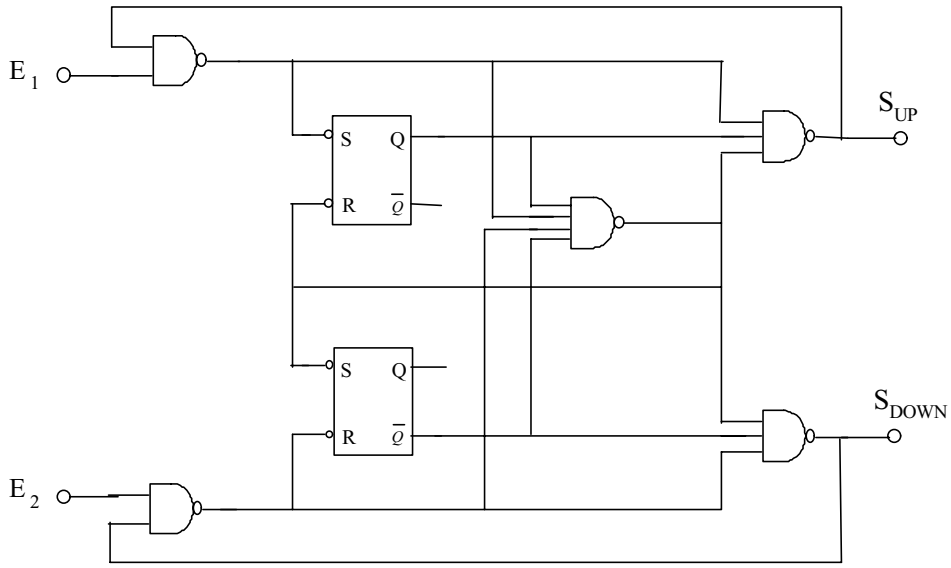

figure 3. 1 : exemple de circuit d'un comparateur phase-fréquence numérique

Un comparateur phase-fréquence génère en sortie deux signaux (S_{up} et S_{down}). Lorsque l'entrée E_1 est en avance de phase sur l'entrée E_2 alors S_{up} est au niveau haut et S_{down} au niveau bas. Lorsque E_1 est en retard de phase sur E_2, on obtient la configuration inverse, c'est-à-dire S_{up} au niveau bas et S_{down} au niveau haut. Lorsque les deux signaux sont synchronisés, S_{up} et S_{down} 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 figure 3. 2 illustre le principe de fonctionnement du PFD :

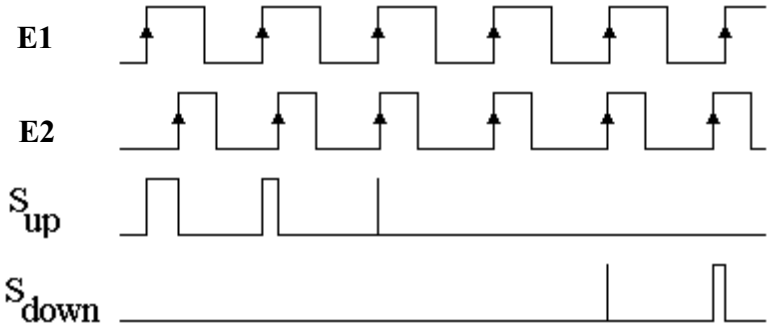

figure 3. 2 : fonctionnement du PFD

b) Modèle de comparateur phase fréquence

Le modèle du comparateur phase-fréquence présenté par la figure 3. 3 a été développé selon une approche fonctionnelle comme expliqué au chapitre précédent. Il est décomposé en

deux parties. La première partie consiste en un processus (*phase_frequency*) dans lequel on calcule une variable intermédiaire '*state*' dont la valeur dépend de l'avance ou du retard du signal E_1 par rapport à E_2. Le processus est activé à chaque détection de front montant de E_1 ou E_2. Dans la deuxième partie du modèle, on génère les deux signaux de sortie S_{UP} et S_{DOWN} conformément au signaux de la figure 3. 2.

Si les deux signaux d'entrée E_1 et E_2 sont synchronisés, la variable *state* prend la valeur 0. Si E_1 est en avance sur E_2, *state* est incrémenté de 1 et prend soit la valeur 1 si elle était à 0 ou 1, soit la valeur 0 si elle était à –1. Si E_1 est en retard sur E_2, *state* est décrémenté de 1 et prend soit la valeur -1 si elle était à 0 ou -1, soit la valeur 0 si elle était à 1.

Les signaux *semaphore_1* et *semaphore_2* sont des signaux de contrôle. *Semaphore_1* (et respectivement *semaphore_2*) prend la valeur de 1 si un front montant de E_1 (respectivement E_2) est détecté et la valeur 0 si un front descendant de E_1 (respectivement E_2) est détecté.

Si *state* est à 1, la sortie S_{UP} prend le niveau *Vhigh*, sinon le niveau *Vlow*. Si *state* est à -1, la sortie S_{DOWN} prend le niveau *Vhigh*, sinon le niveau *Vlow*.

```
entity phase_frequency_detector is
      generic (Vhigh, Vlow, trise, tfall : real);
      port (terminal Tinput_1, Tinput_2, Toutput_up, Toutput_down : electrical);
end entity phase_frequency_detector;

architecture behavior of phase_frequency_detector is
      constant Vthreshold : real :=0.5*(Vhigh+Vlow);
      quantity Vin_1 across Iin_1 through Tinput_1 to electrical_ground;
      quantity Vin_2 across Iin_2 through Tinput_2 to electrical_ground;
      quantity Vout_up across Iout_up through Toutput_up to electrical_ground;
      quantity Vout_down across Iout_down through Toutput_down to electrical_ground;
      signal state : real := 0.0;
      signal semaphore_1      : bit := '0';
      signal semaphore_2      : bit := '0';
begin
Iin_1 == 0.0;
Iin_2 == 0.0;
phase_frequency : process
begin
          -- détection des variables d'entrée : test d'avance ou de retard de phase --
```

```
                    -- et calcul des paramètres de sortie : l'état 'state' --

    Wait until (Vin_1'above(Vthreshold)'event) or (Vin_2'above(Vthreshold)'event);
                                    -- E1 et E2 synchronisés --
    If   (Vin_1'above(Vthreshold) = true) and (semaphore_1 = '0') and
         (Vin_2'above(Vthreshold) = true) and (semaphore_2 = '0') then semaphore_1 <= '1';
         semaphore_2 <= '1';
         state <= 0.0;
    else
                                    -- E1 en avance sur E2 --
         if (Vin_1'above(Vthreshold) = true) and (semaphore_1 = '0') then semaphore_1 <= '1';
              if ((state = 0.0) or (state = 1.0)) then state <= 1.0;
              else state <= 0.0;
              end if;
         end if;
                                    -- E1 en retard sur E2 --
         if (Vin_2'above(Vthreshold) = true) and (semaphore_2 = '0') then semaphore_2 <= '1';
              if ((state = 0.0) or (state = -1.0)) then state <= -1.0;
              else state <= 0.0;
              end if;
         end if;
    end if;

   -- détection de front descendant et remise à zéro de semaphore_1 ou semaphore_2 --
    if (Vin_1'above(Vthreshold) = false) then semaphore_1 <= '0';
    end if;
    if (Vin_2'above(Vthreshold) = false) then semaphore_2 <= '0';
    end if;
end process phase_frequency;

                -- génération des deux signaux de sortie 'Vout_up' et 'Vout_down' --
      if state > 0.0 use Vout_up == Vhigh*state'ramp(trise, tfall);
      else Vout_up == Vlow;
      end use;
      if state < 0.0 use Vout_down == Vhigh*state'ramp(trise, tfall);
      else Vout_down == Vlow;
      end use;
End architecture behavior;
```

figure 3. 3 : modèle du comparateur phase fréquence

Un exemple de résultat de simulation du modèle avec ADVanceMS de Mentor Graphics est illustré par les courbes de la figure 3. 4 où sont présentées les deux entrées E_1 et E_2 et les sorties S_{UP} et S_{DOWN} :

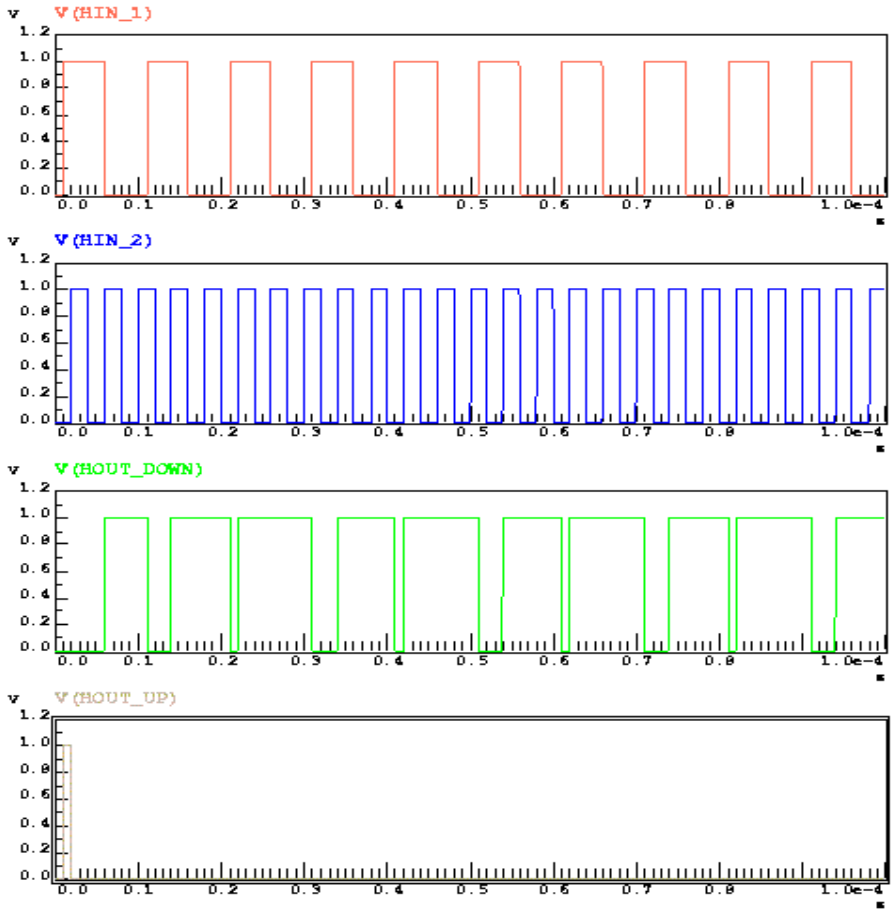

figure 3. 4: simulation du modèle du comparateur phase-fréquence

II.1.2 Pompe de charge

a) Principe de fonctionnement

La pompe de charge est un circuit qui est généralement placé après le comparateur phase-fréquence pour piloter par exemple un oscillateur contrôlé en tension (VCO) dans un système de boucle à verrouillage de phase (PLL). Elle permet de combiner les deux signaux de sortie du comparateur S_{up} et S_{down} en un seul signal, ce qui est généralement nécessaire pour piloter le VCO. L'association de ces deux circuits permet ainsi d'avoir une plage de capture de la PLL limitée à la plage de variation de la fréquence du VCO et une erreur de phase statique nulle lorsque la boucle est verrouillée. Un exemple de circuit de pompe de charge est présenté par la figure 3. 5 :

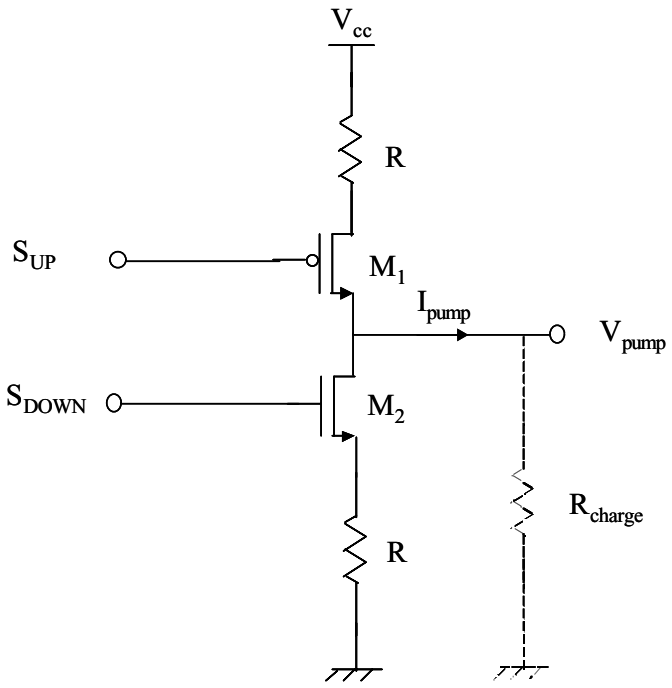

figure 3. 5 : exemple de circuit de pompe de charge

Le signe du courant de sortie I_{pump} de la pompe de charge varie selon l'état des entrées S_{up} et S_{down}. Lorsque S_{up} est à son niveau haut, I_{pump} est positif et lorsque S_{down} est à son niveau haut, I_{pump} est négatif. La figure 3. 6 illustre le comportement du bloc pompe de charge :

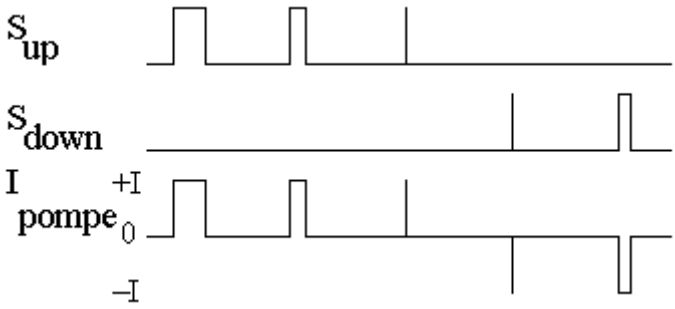

figure 3. 6 : fonctionnement de la pompe de charge

b) Modèle de la pompe de charge

Le modèle de la pompe de charge est développé selon une approche fonctionnelle. Le modèle est décomposé en deux parties. Dans la première partie, on fait un test sur l'état des deux entrées V_{UP} et V_{DOWN} et on génère deux signaux intermédiaires *'semaphore_1'* et '*semaphore_2'*. Le processus qui suit est activé à chaque front de l'un des deux signaux V_{UP}

ou V_{DOWN}. Si un front montant de V_{UP} (respectivement V_{DOWN}) est détecté, *Semaphore_1* (respectivement *semaphore_2*) prend la valeur de 1. Par contre si un front descendant de V_{UP} (respectivement V_{DOWN}) est détecté, *Semaphore_1* (respectivement *semaphore_2*) prend la valeur de 0.

Dans la deuxième partie du modèle, on génère le courant de sortie I_{PUMP} selon l'état de *'semaphore_1'* et '*semaphore_2'*.

```
entity charge_pump is
    generic (Vhigh, Vlow, trise, tfall, current_amplitude : real);
    port (terminal Tinput_up, Tinput_down, Toutput : electrical);
end entity charge_pump;

architecture behavior of charge_pump is
    constant Vthreshold : real := 0.5*(Vhigh+Vlow);
    quantity Vup across Iup through Tinput_up to electrical_ground;
    quantity Vdown across Idown through Tinput_down to electrical_ground;
    quantity Vout across Iout through Toutput to electrical_ground;
    signal semaphore_1 : real := 0.0;
    signal semaphore_2 : real := 0.0;
    signal difference_semaphore : real := 0.0;
begin
Iup == 0.0;
Idown == 0.0;
pump : process
begin
    --détection des variables d'entrée : test de l'état des entrées 'Vup' et 'Vdown' --
    -- et calcul des paramètres de sortie : génération des deux signaux intermédiaires
                    'semaphore_1' et 'semaphore_2' --
    wait until (Vup'above(Vthreshold)'event) or (Vdown'above(Vthreshold)'event);

                    -- détection d'un front montant de 'Vup' --
    if (Vup'above(Vthreshold) = true) and (semaphore_1 = 0.0) then semaphore_1 <= 1.0;
    end if;

                    -- détection d'un front descendant de 'Vup' --
    if (Vup'above(Vthreshold) = false) then semaphore_1 <= 0.0;
    end if;

                    -- détection d'un front montant de 'Vdown' --
```

```
    if (Vdown'above(Vthreshold) = true) and (semaphore_2 = 0.0) then semaphore_2 <= 1.0;
    end if;

                        -- détection d'un front descendant de 'Vup' --
    if (Vdown'above(Vthreshold) = false) then semaphore_2 <= 0.0;
    end if;
    end process pump;
    difference_semaphore <= semaphore_1 – semaphore_2;

                        -- génération du courant de sortie 'Iout'
Iout/current_amplitude == - difference_semaphore'ramp(trise, tfall);
end architecture behavior;
```

figure 3. 7 : modèle de la pompe de charge

Des courbes de simulation du modèle de la pompe de charge avec ADVanceMS sont données par la figure 3. 8. Il s'agit des entrées V_{UP} et V_{DOWN} et de la sortie I_{PUMP} débitant sur une résistance de 1kΩ.

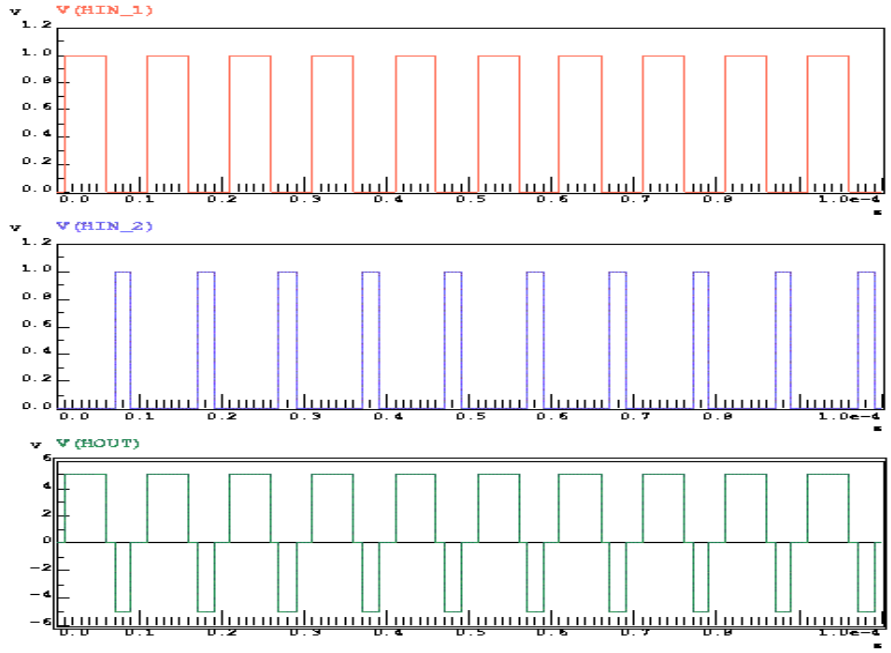

figure 3. 8 : simulation du modèle de la pompe de charge

II.2 Filtres passe bas

Pour développer des modèles de filtre, nous avons adopté une démarche schématique.

Nous présentons le modèle d'un filtre passe bas qui va être utilisé par la suite dans les modèles de PLL et de synthétiseurs de fréquence. Son schéma électrique est le suivant :

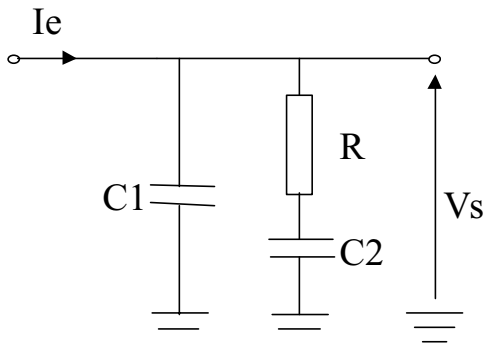

figure 3. 9 : filtre passif passe bas

Pour modéliser le filtre, on définit le courant et la tension à chaque nœud de son schéma électrique en appliquant les lois de Kirchhoff :

```
entity loop_filter is
      generic (C1, C2, R2 : real);
      port (terminal Tinput, Toutput : elecrical);
end entity loop_filter;

architecture behavior of loop_filter is

                  -- détection du courant d'entrée
      quantity Vin across Iin through Tinput to electrical_ground;
      quantity Vout across Iout through Toutput to electrical_ground;
      quantity I_C1 : real;
      quantity V_C1 : real;
begin
       -- définition des tensions et des courants pour les deux
                  -- nœuds du schéma électrique
      I_C1 == C_1*Vout'dot;
      Iin – I_C1 + Iout == C_2*V_C2'dot;
      Vout == V_C2 + R_2*(Iin – I_C1 + Iout);
      Vin == Vout;
end architecture behavior;
```

figure 3. 10 : modèle du filtre passe bas

Les résultats d'une simulation fréquentielle du modèle (diagramme de Bode) sont donnés par la figure 3. 11 :

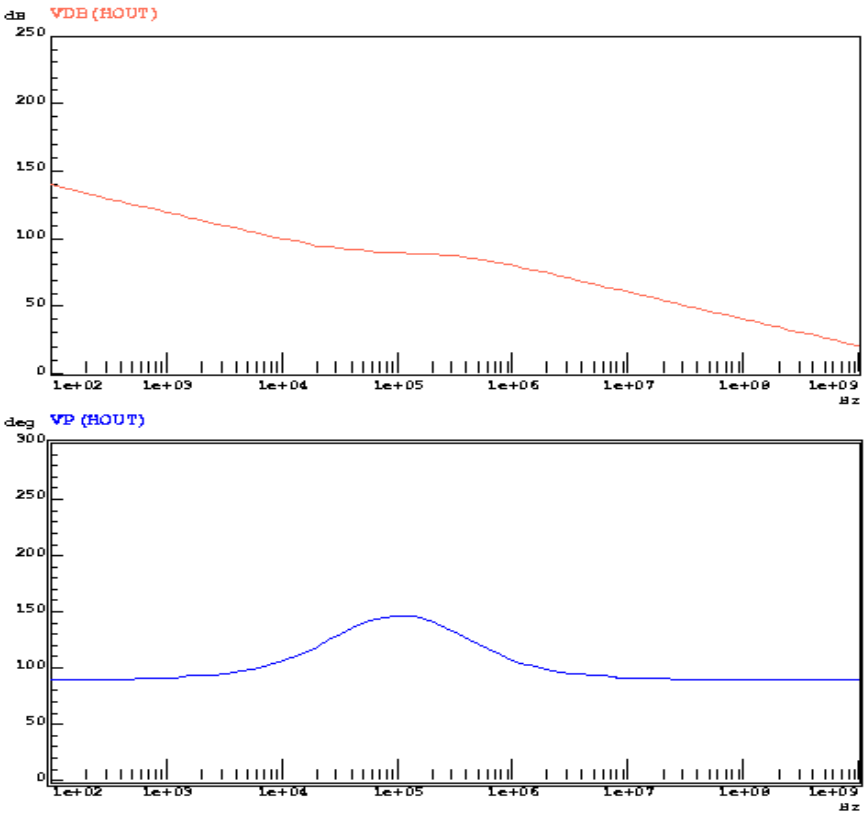

figure 3. 11 : diagrammes de Bode issus de la simulation du modèle du filtre

II.3 Diviseurs de fréquence

Les diviseurs de fréquence sont des circuits qui divisent la fréquence du signal d'entrée d'un rapport entier déterminé. Il s'agit généralement de circuits numériques utilisant des bascules et des multiplexeurs. Notre bibliothèque contient deux types de diviseurs : des diviseurs par N classiques dont le rapport de division N est constant et des diviseurs N/M qui peuvent changer de rapport de division (de N à M) suivant l'état d'un signal de commande extérieur.

II.3.1 Diviseur N

Nous avons développé deux méthodes pour modéliser les diviseurs de fréquence. Le premier principe de modélisation consiste à détecter la période d'entrée et de la multiplier par le rapport de division N. Le modèle développé contient alors dans sa première partie le processus de détection de période dont le principe a été exposé dans le deuxième chapitre.

Dans la deuxième partie du modèle, on calcule la période de sortie. La dernière partie du modèle permet de générer le signal de sortie.

Dans le modèle du diviseur N présenté par la figure 3. 12, le signal de sortie est un signal carré de rapport cyclique ajustable.

```
entity frequency_divider is
      generic (Vhigh, Vlow, trise, tfall, divisor, cyclic_ratio : real; delay, initial_period : time);
      port ( terminal Tinput, Toutput : electrical);
end entity frequency_divider ;

architecture behavior of frequency_divider is
      constant Vthreshold : real := 0.5*(Vhigh+Vlow);
      quantity Vin across Tinput to electrical_ground;
      quantity Vout across Iout through Toutput to electrical_ground;
      signal time_conv     : time := 0fs;
      signal thigh         : time := cyclic_ratio*initial_period;
      signal period                : time := initial_period;
      signal tbegin                : time := 0fs;
      signal Semaphore_1           : real := 1.0;
      signal Semaphore_2           : real := Vlow;
begin
                              -- détection de la période d'entrée --
calcul_periode : process
begin
       wait until Vin'above(Vthreshold) = true;
       time_conv <= real2time(now);
       if tbegin = 0fs then  tbegin <= time_conv;
                             thigh <= cyclic_ratio*initial_period;

                              -- calcul de la periode de sortie --
                             period <= initial_period;
       else          period <= divisor*(time_conv - tbegin) after (delay);
                     thigh <= divisor*cyclic_ratio*(time_conv - tbegin) after (delay);
                     tbegin <= time_conv;
       end if;
end process calcul_periode;
 -- génération d'un signal de rapport cyclique 0.5 et de fréquence égale la fréquence
                                                de sortie--
Semaphore_1 <= -Semaphore_1 after (0.5*period);
```

```
-- génération du signal de sortie, de rapport cyclique ajustable, synchronisé sur le signal 'semaphore_1' --
output_generation : process
begin
        wait until Semaphore_1'event;
        if Semaphore_1 = -1.0 then Semaphore_2 <= Vhigh, Vlow after (thigh);
        end if;
end process output_generation;
Vout == Semaphore_2'ramp(trise, tfall);
end architecture behavior;
```

figure 3. 12 : modèle du diviseur de fréquence par N

Un deuxième principe de modélisation 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. C'est le principe des diviseurs de fréquence utilisant des bascules D. Ce principe est illustré par la figure 3. 13 :

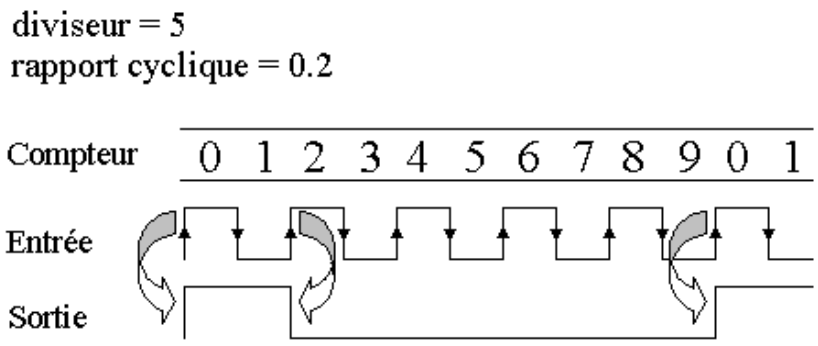

figure 3. 13 : principe des diviseurs de fréquence utilisant des bascules D

Un exemple de modèle traduisant ce principe de modélisation est présenté par la figure 3. 14 :

```
entity diviseur_de_frequence_avec_compteur is
        generic (Vh, Vb, tmontee, tdescente, rapport_division, rapport_cyclique : real);
        port ( terminal horloge_entree, horloge_sortie : electrical);
end entity diviseur_de_frequence_avec_compteur ;

architecture comportemental of diviseur_de_frequence_avec_compteur is
        constant Vmilieu :real := 0.5*(Vh+Vb);
        quantity Ventree across Ientree through horloge_entree to electrical_ground;
```

```
        quantity Vsortie across Isortie through horloge_sortie to electrical_ground;
        signal compteur : real :=-1.0;
        signal intermediaire : real := Vb;
begin
Ientree == 0.0;
front : process
begin
        -- détection des variables d'entrée : détection du front montant du signal d'entrée --
          wait on Ventree'above(Vmilieu);
                                -- incrémentation du compteur --
          compteur <= compteur + 1.0;
                        -- calcul d'un signal intermédiaire qui définira l'état de la sortie --
          if (compteur<2.0*(rapport_division)*rapport_cyclique) then intermediaire <= Vb;
          else intermediaire <= Vh;
          end if;
                                -- remise à zéro du compteur --
          if (2.0*(rapport_division-1.0)<compteur) then compteur <= 0.0;
          end if;


          end process front;
                                -- génération du signal de sortie --
Vsortie == intermediaire'ramp(tmontee,tdescente);
end architecture comportemental;
```

figure 3. 14 : modèle de diviseur par N utilisant le principe de comptage des fronts

Pour un rapport de division égal à 4 et un rapport cyclique de 0.25, la figure 3. 15 donne un exemple de résultat de simulation avec ADVanceMS :

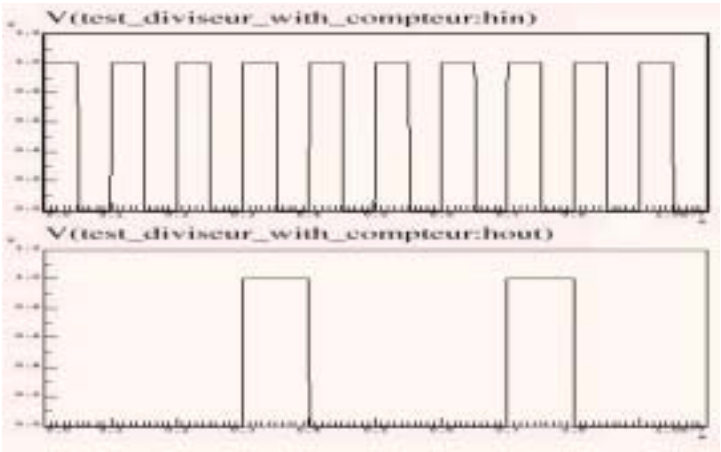

figure 3. 15 : simulation du diviseur de fréquence

II.3.2 Diviseur N/M

a) Principe de fonctionnement

Le diviseur N/M est commandé par un signal extérieur qui détermine le rapport de division N ou M. La figure 3. 16 montre le chronogramme d'un diviseur N/N+1. Lorsque l'entrée de commande est au niveau haut, le rapport de division est égal à N+1 ; lorsqu'elle est au niveau bas, le rapport est égal à N.

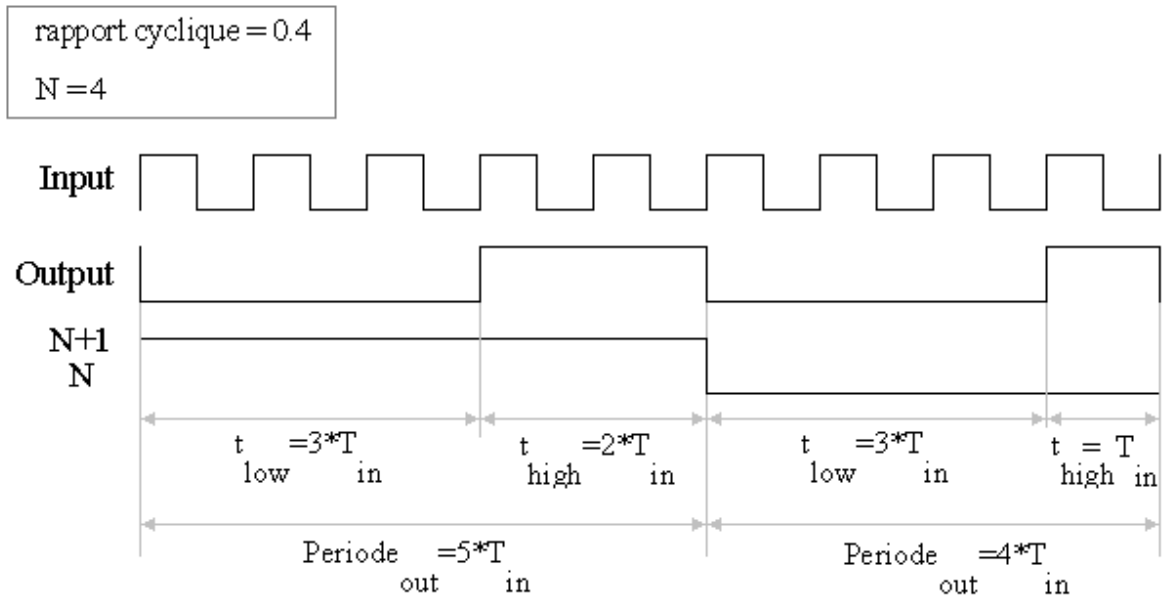

figure 3. 16 : chronogramme d'un diviseur de fréquence N/N+1

b) Modèle du diviseur N/M

Les modèles développés pour les diviseurs N/M utilisent les deux principes de modélisation exposés pour les diviseurs N. Une seule partie de test est ajoutée et permet de fixer le rapport de division (N ou M) selon l'état de l'entrée de commande '*carry*' :

```
-- determination du rapport de division --
if carry = '1' use divisor == divisor_N;
else divisor == divisor_M;
end use;
```

figure 3. 17 : détermination du rapport de division

La figure 3. 18 illustre l'influence d'un changement du rapport de division sur la sortie d'un diviseur 19/20 :

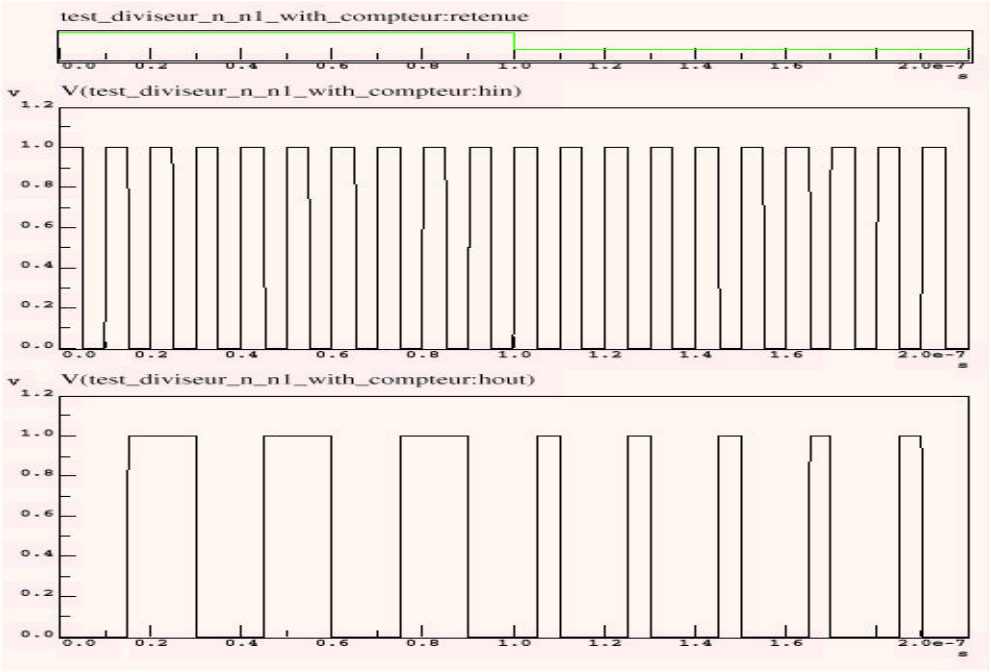

figure 3. 18 : simulation du modèle du diviseur de fréquence N/N+1

II.3.3 Accumulateur

L'accumulateur est le circuit qui génère le signal permettant de commander le diviseur N/M et d'effectuer le choix du rapport de division. La figure 3. 19 présente le schéma bloc de l'accumulateur modélisé :

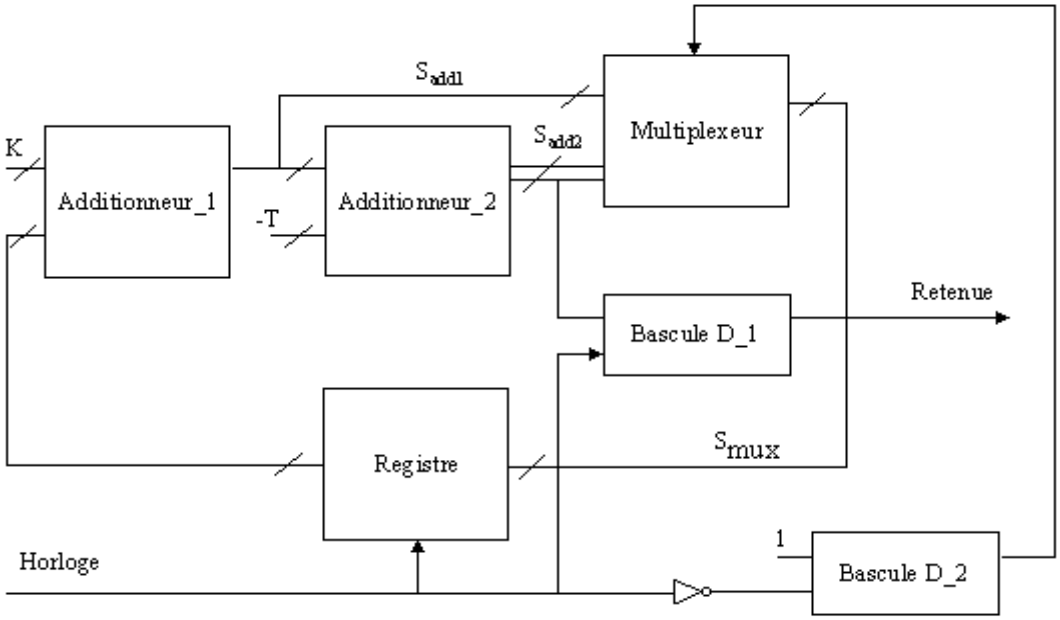

figure 3. 19 : schéma bloc de l'accumulateur

a) Principe de fonctionnement

Il s'agit d'un circuit numérique possédant trois entrées T, K et une horloge. L'entrée T fixe la période du signal de sortie (égale à T fois la période de l'horloge) et K le nombre de fronts montants par période de sortie.

L'accumulateur est constitué de 2 additionneurs, un multiplexeur, un registre, un inverseur et deux bascules D. Le premier additionneur additionne la valeur de K avec la sortie du registre. Le résultat sera additionné avec la valeur -T. Les sorties du premier et deuxième additionneur S_{add1} et S_{add2} sont multiplexées : si S_{add2} est positive, elle sera récupérée à la sortie du multiplexeur, sinon on récupère S_{add1}. 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 S_{add2} est dirigé vers la bascule D_1 et sera le signal de commande du diviseur N/M. Si S_{add2} est positive, il est égal à 1, sinon il vaut 0.

Pour mieux saisir le fonctionnement de l'accumulateur, considérons l'exemple où l'on a T = 20 et K = 4. L'additionneur 1 additionne la valeur de K avec la valeur du registre, initialement égale à 0. On obtient à sa sortie la valeur 4 et on retrouve alors à la sortie de l'additionneur 2 la valeur –16. Puisque cette dernière valeur est négative, on retrouve à la sortie du multiplexeur la valeur S_{add1} = 4.et la retenue est égale à 0. La sortie du multiplexeur est mémorisée dans le registre et au prochain coup d'horloge, on la retrouve à l'entrée de

l'additionneur 1. On aura alors S_{add1} égale à 4 + 4 = 8 et le cycle recommence pour retrouver à chaque coup d'horloge les valeurs suivantes : 4, 8, 12, 16, 0, 4, 8, etc. Respectivement, la retenue prend les valeurs : 0, 0, 0, 0, 1, 0, 0, etc. Au bout de T coups d'horloge, la retenue passe 4 fois par 1 et 16 fois par 0. La figure 3. 20 donne le chronogramme de fonctionnement interne de l'accumulateur :

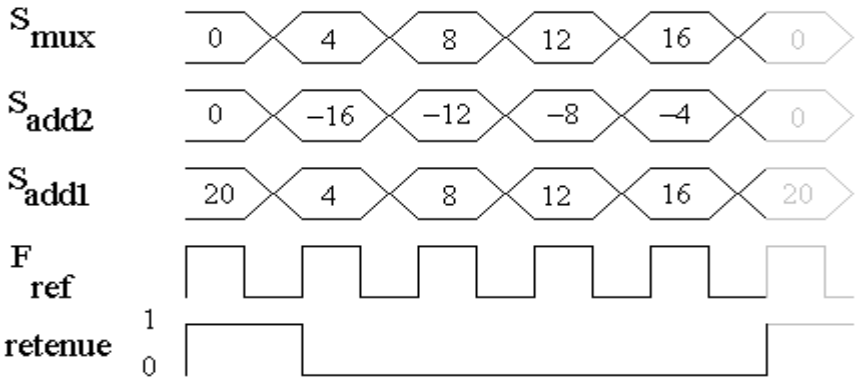

figure 3. 20 : Chronogramme de fonctionnement interne de l'accumulateur

b) Modèle de l'accumulateur

Dans le modèle de l'accumulateur, T et K sont considérés comme paramètres génériques. L'entrée électrique est l'horloge qui synchronise le fonctionnement du circuit et la sortie *'carry'* est un signal numérique qui commande le diviseur N/M. Le modèle reprend le principe de fonctionnement de l'accumulateur et remplace les additionneurs par un simple opérateur d'addition et les multiplexeurs par un opérateur de condition (*if then*). Le processus '*accumu*' développé est activé à chaque front montant du signal d'horloge.

```
entity accumulator is
        generic (T, K, Vhigh, Vlow : real);
        port (terminal Tinput : electrical; signal carry : out bit);
end entity accumulator;

architecture behavior of accumulator is
        constant Vth : real := 0.5*(Vhigh+Vlow);
        signal counter_1 : real := 0.0;
        signal mux : real := 0.0;
        quantity counter_2 : real := -20.0;
        quantity Vin across Tinput to electrical_ground;
begin
```

```
accumu : process
begin
                    -- premier additionneur --
    counter_1 <= K + mux;

      -- multiplexage et détermination de la sortie 'carry' --
    if counter_2 >= 0.0 then mux <= counter_2;
    carry <= '0';
    else mux <= counter_1;
    carry <= '1';
    end if;
    wait on Vin'above(Vth);
end process accumu;

                    -- deuxième additionneur --
counter_2 == counter_1 – T;
end architecture behavior;
```

figure 3. 21 : modèle de l'accumulateur

Pour une période d'horloge égale à 10 ns, T égal à 20 et K égal à 4, nous obtenons le résultat de simulation présenté par la figure 3. 22. Il vérifie le chronogramme de fonctionnement de l'accumulateur de la figure 3. 20 :

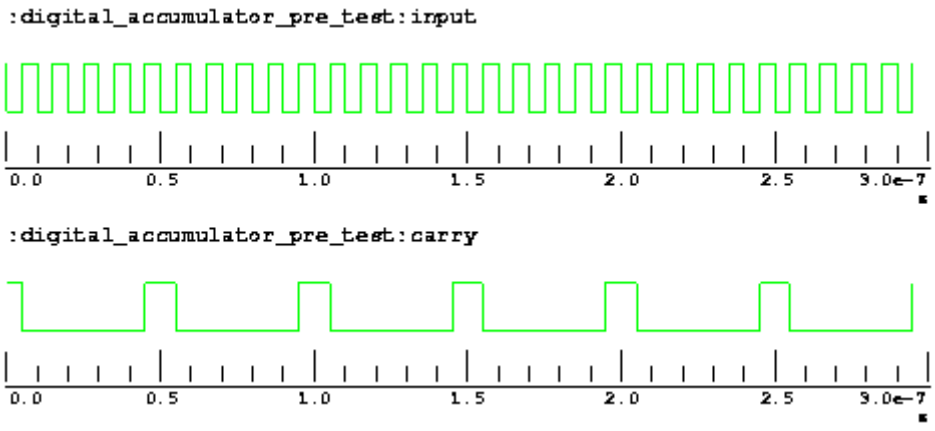

figure 3. 22 : simulation de l'accumulateur

II.3.4 Diviseur fractionnaire

a) Principe de fonctionnement

Un diviseur de fréquence fractionnaire permet de diviser la fréquence du signal d'entrée non seulement par un entier mais aussi 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.

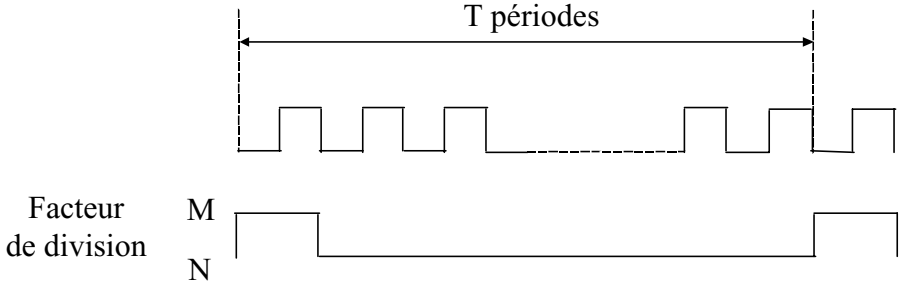

figure 3. 23 : principe de la division fractionnaire

La figure 3. 23 illustre le principe de la division fractionnaire : la fréquence du signal d'entrée est divisée une fois par M toutes les T périodes et par N le reste du temps, c'est-à-dire pendant T-1 périodes. Le facteur de division fractionnaire est donc donné par :

$$N_{FRAC} = M\left(\frac{1}{T}\right) + N\left(\frac{T-1}{T}\right) = N + \frac{M-N}{T} \qquad (3.\ 1)$$

Si on divise non pas une fois mais K fois par M, on divise par conséquent T-K fois par N. Le facteur de division fractionnaire devient alors :

$$N_{FRAC} = M\left(\frac{K}{T}\right) + N\left(\frac{T-K}{T}\right) = N + \frac{K(M-N)}{T} \qquad (3.\ 2)$$

K est le paramètre qui permet de fixer le rapport de division fractionnaire. Il est compris entre 0 et T. N_{frac} peut varier par conséquent entre N et M.

b) Modèle du diviseur fractionnaire

Un modèle de diviseur fractionnaire est obtenu en associant l'accumulateur au diviseur N/M. Dans le modèle de la figure 3. 24, on fait appel au modèle de l'accumulateur et au

diviseur N/N+1. Il s'agit donc d'un modèle structurel. En agissant sur les paramètres T et K, on peut définir le rapport de division fractionnaire.

```
entity fractional_frequency_divider is
    generic (T, K, divisor_N, Vhigh, Vlow, trise, tfall, cyclic_ratio : real; delay, initial_period : time);
    port ( terminal Tinput_1, Tinput_2, Toutput : electrical);
end entity fractional_frequency_divider;

architecture behavior of fractional_frequency_divider is
    signal carry : bit := '0';
begin
                        -- appel au modèle de l'accumulateur --
accumulator : entity work.accumulator(behavior)
    generic map (T => T, K => K, Vhigh => Vhigh, Vlow => Vlow)
    port map (Tinput => Tinput_1, carry => carry);

                        -- appel au modèle du diviseur --
divider: entity work.frequency_divider_N_N_plus_1(behavior)
    generic map (Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall, divisor_N=>divisor_N,
    cyclic_ratio=>cyclic_ratio, delay=>delay, initial_period=>initial_period)
    port map ( Tinput=>Tinput_2, Toutput=>Toutput, carry=>carry );
end architecture behavior;
```

figure 3. 24 : modèle structurel du diviseur fractionnaire

II.4 Circuits oscillateurs

Notre bibliothèque contient des modèles pour deux types d'oscillateurs : les oscillateurs contrôlés en tension et les oscillateurs synchrones.

II.4.1 Oscillateur contrôlé en tension

On distingue généralement deux grandes familles d'oscillateurs contrôlés en tension (Voltage Controlled Oscillator ou VCO) : les oscillateurs à relaxation parmi lesquels on trouve les oscillateurs astables et en anneau, et les oscillateurs à réseaux LC parmi lesquels on trouve les oscillateurs harmoniques, les oscillateurs à résistances négatives ou les oscillateurs à déphasage.

Le VCO présenté dans le chapitre précédent est un oscillateur astable dont le principe consiste à charger et décharger une capacité par un courant constant. La fréquence de sortie

est alors proportionnelle au rapport du courant sur la valeur de la capacité. Nous avions suivi une approche schématique pour ce modèle. Ici, nous présentons un modèle générique de VCO à sortie carré, développé selon une approche fonctionnelle.

c) Principe de fonctionnement

En l'absence de signal d'entrée, un oscillateur contrôlé en tension oscille à sa fréquence propre f_0. En appliquant une tension d'entrée V_{in}, la fréquence de sortie varie proportionnellement à cette tension, suivant l'expression suivante :

$$f_{vco} = f_0 + K_{vco}.V_{in} \qquad (3.\ 3)$$

K_{VCO} est la sensibilité du VCO exprimée en Hz/V.

d) Modèle du VCO

Dans le modèle de la figure 3. 25, la fréquence propre du VCO est considérée comme paramètre générique. Le modèle est décomposé en trois parties : on commence par détecter la valeur de la tension d'entrée qui va permettre de calculer la période de sortie en appliquant l'expression (3. 3). Enfin, on génère le signal de sortie en respectant le rapport cyclique donné comme paramètre générique :

```
entity VCO is
      generic (Vhigh, Vlow, trise, tfall, cycle_ratio, vco_gain : real;
                  delay, vco_period : time);
      port (terminal Tinput, Toutput : electrical);
end entity VCO;

architecture behavior of VCO is

                        -- détection de la tension d'entrée --
      quantity Vin across Tinput to electrical_ground;
      quantity Vout across Iout through Toutput to electrical_ground;
      signal tlow : time := vco_period*(1.0-cyclic_ratio);
      signal period : time := vco_period;
      signal semaphore_1 : real := 1.0;
      signal semaphore_2 : real := Vlow;
begin
semaphore_1 <= -semaphore_1 after (0.5*period);
```

```
output_period : process
begin
    wait until semaphore_1'event;

    -- calcul des paramètres de sortie : calcul de la période de sortie --
    period <= real2time(1.0/((1.0/time2real(vco_period))+Vin*vco_gain));

                    -- génération du signal de sortie --
    tlow <= (1.0-cyclic_ratio)*real2time(1.0/(1.0/time2real(vco_period)));
    if semaphore_1 = -1.0 then semaphore_2 <= Vlow, Vhigh after (tlow);
    end if;
end process output_period;
Vout == semaphore_2'ramp(trise, tfall);
end architecture behavior;
```

figure 3. 25 : modèle du VCO

II.4.2 Oscillateur synchrone

e) Présentation des oscillateurs synchrones :

Un oscillateur est un circuit qui délivre un signal périodique de fréquence propre f_0 en l'absence de signal d'entrée. Il existe des oscillateurs particuliers disposant d'une entrée. En appliquant à cette entrée un signal périodique de fréquence f_e proche de la fréquence propre f_0, ils se mettent à osciller à la fréquence f_e. C'est le phénomène de synchronisation des oscillateurs, longtemps utilisé dans des applications radio fréquences (RF). De tels oscillateurs sont appelés oscillateurs synchrones (OS).

En 1985, V. UZUNOGLU a proposé un oscillateur synchrone discret. Il s'agit d'un oscillateur COLPITTS modifié qui présente les propriétés suivantes [Bad00] :

- Il est capable de se synchroniser sur des harmoniques de sa fréquence d'oscillation (synchronisation harmonique), sur une fréquence proche de sa fréquence propre (synchronisation sur le fondamental) et sur des fréquences dont il est harmonique (synchronisation sous harmonique) ;

- Le temps d'acquisition est largement inférieur à celui d'une boucle à verrouillage de phase ;

- Il se comporte comme un filtre passe bande dont la bande passante est ajustée par l'amplitude du signal le synchronisant. Ainsi, si le rapport signal sur bruit du signal d'entrée décroît, l'OS réduit sa plage de capture de façon à maintenir constant le rapport signal sur bruit du signal de sortie ;

- Dans la plage de capture, l'OS recopie le bruit de phase du signal qui le synchronise au facteur de multiplication ou de division près ;

  - L'OS est un système à mémoire : il reste synchronisé un certain temps sur la fréquence d'une porteuse en l'absence de celle-ci ;

Ces nombreuses propriétés permettent d'envisager l'utilisation des OS dans de nombreuses applications tel que la synthèse de fréquence.

Nous allons étudier deux types d'oscillateurs synchrones : les oscillateurs COLPITTS et les oscillateurs à résistance négative.

e.i Oscillateur synchrone COLPITTS

Le schéma de principe d'un oscillateur synchrone COLPITTS est présenté par la figure 3.26. Il existe trois façons pour le synchroniser. On peut placer un générateur de courant de synchronisation i_{sync} en parallèle soit avec C_1, soit avec C_2, soit avec L.

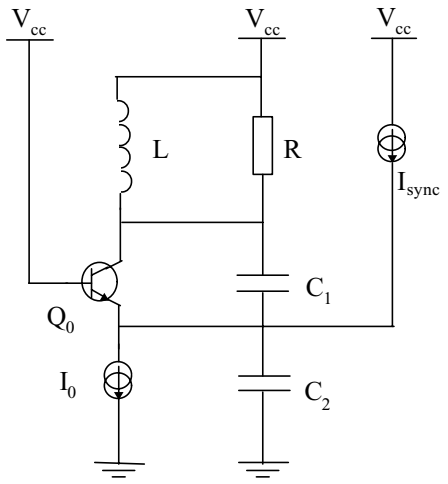

figure 3.26 : schéma de principe de l'oscillateur synchrone COLPITTS

Pour l'oscillateur de la figure 3.26, la synchronisation se fait en appliquant le générateur I_{sync} en parallèle avec la capacité C_2. La fréquence propre de l'oscillateur est donnée par l'expression suivante :

$$f_0 = \frac{1}{2\pi} \frac{1}{\sqrt{L \frac{C_1 C_2}{C_1 + C_2}}} \qquad (3.4)$$

e.ii Oscillateur synchrone à résistance négative

Le schéma de principe d'un oscillateur synchrone à résistance négative est donné par la figure 3. 27 :

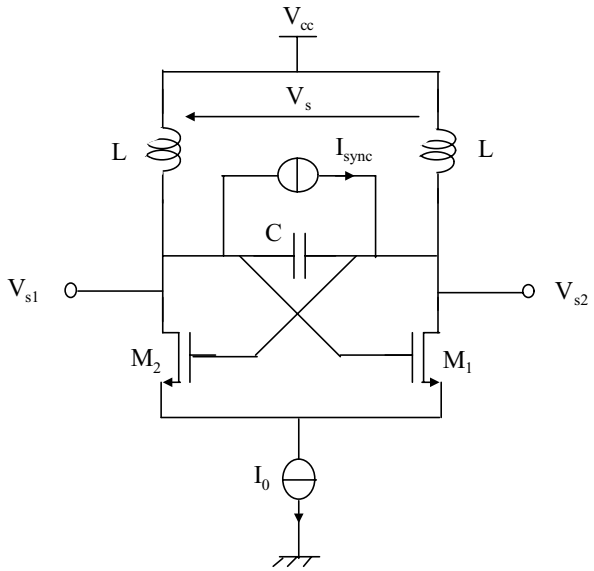

figure 3. 27 : schéma de principe de l'OS à résistance négative

La synchronisation se fait aux bornes de la capacité C, ce qui assure une plus grande plage de synchronisation [Bad00]. La résistance négative est réalisée par la paire croisée NMOS. Celle ci compense les pertes du réseau LC, assurant ainsi l'oscillation du système.

f) Modélisation des oscillateurs synchrones

Nous avons développé dans notre bibliothèque RF des modèles d'oscillateurs synchrones de type COLPITTS et à résistance négative. Ces modèles ont été écrits en s'appuyant sur la théorie d'Huntoon et Weiss consernant les oscillateurs synchrones

[HuW47]. Nous allons commencer par rappeler les principaux résultats obtenus en appliquant cette théorie à nos oscillateurs synchrones[Bad00].

f.i Modélisation de l'oscillateur COLPITTS

Le schéma électrique simplifié de l'oscillateur COLPITTS modélisé est le suivant :

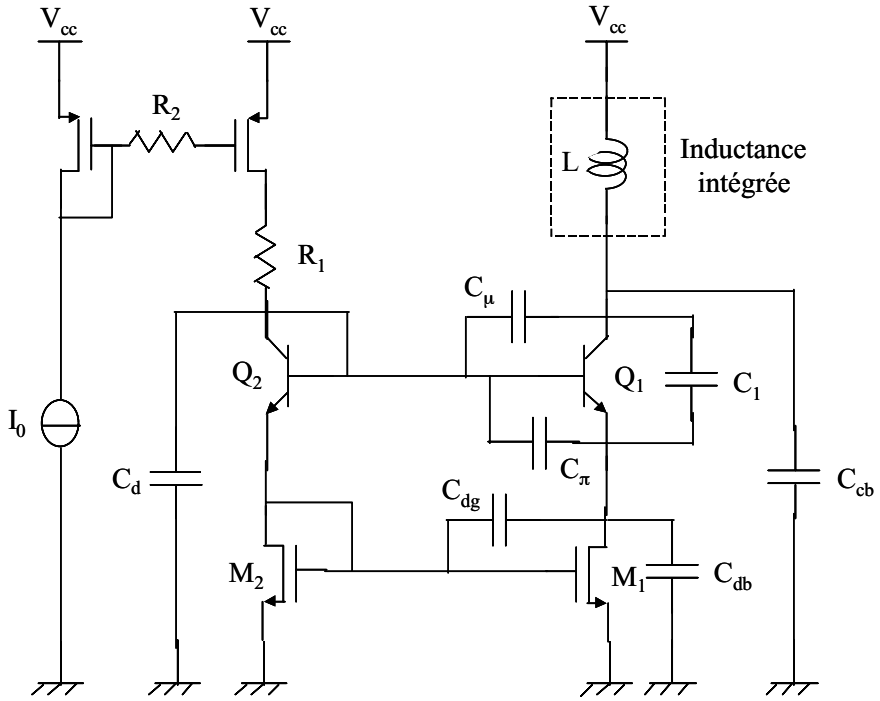

figure 3.28 : oscillateur COLPITTS base commune

L'oscillateur synchrone est caractérisé par sa plage de synchronisation qui est fonction de l'amplitude du signal de synchronisation. Il est également important d'évaluer sa caractéristique de bruit de phase.

L'application de la théorie d'Huntoon et Weiss à l'oscillateur nous permet d'exprimer sa fréquence propre f_0 en fonction des différents paramètres du circuit :

$$f_0 = \frac{1}{2\pi}\sqrt{\frac{1+\left(\frac{L.g_c}{C_2^{'}+C_1}\right)\left(g_m+g_\pi\right)}{L\left(C_\mu+\frac{C_1.C_2^{'}}{C_1+C_2^{'}}\right)}} \qquad (3.\ 5)$$

Avec $C_2^{'} = C_2 + C_{dg} + C_{db} + C_\pi$

et : C_μ : capacité base-collecteur

C_π : Capacité base-émetteur

C_{dg} : capacité drain-grille

C_{db} : capacité drain-bulk

g_m : transconductance du transistor

g_π : transconductance base-émetteur

g_c : transconductance base-collecteur

Dans la théorie d'Huntoon et Weiss, la source de synchronisation est modélisée par une petite admittance *dy* de la forme : $dy = g + jb$, avec g et b respectivement les parties réelle et imaginaire de l'admittance. Lorsque l'oscillateur est synchronisé, l'expression de la fréquence d'oscillation est donnée par l'expression suivante, tenant compte de g :

$$f = \frac{1}{2\pi}\sqrt{\frac{1+\left(\frac{L\,g_c}{C_2'' + C_1}\right)\left(g_m + g_\pi'\right)}{L\left(C_\mu + \frac{C_1\,C_2''}{C_1 + C_2''}\right)}} \qquad (3.\,6)$$

Avec : $g_\pi' = g_\pi + g$

et $C_2'' = C_2' + \frac{b}{\omega}$

On définit les facteurs d'élasticité E_F et E_A comme étant respectivement les variations de fréquence et d'amplitude des oscillations de sortie.

Le facteur E_F peut s'écrire sous la forme : $E_F = F_g + jF_b$ avec F_g la dérivée de la fréquence f par rapport à g et F_b la dérivée de f par rapport à b. On obtient alors :

$$F_g = \frac{1}{8\pi^2}\frac{1}{f_0}\frac{g_c}{C_\mu\left(C_2' + C_1\right) + C_2'\,C_1} \qquad (3.\,7)$$

$$F_b = \frac{1}{4\,\pi}\left(\frac{1}{C_1 + C_2'}\right)^2\left[\frac{L\,g_c\left(g_m + g_\pi'\right)}{1+\frac{L\,g_c\left(g_m + g_\pi'\right)}{C_1 + C_2'}} + \frac{C_1^2}{C_\mu + \frac{C_1\,C_2'}{C_1 + C_2'}}\right] \qquad (3.\,8)$$

F_g et F_b sont appelés facteurs de compliance.

La largeur de la plage de synchronisation Δf est définie par l'expression suivante :

$$\Delta f = 2\frac{I_{sync}}{V_0}\sqrt{F_b^2 + F_g^2} \qquad (3.\ 9)$$

avec V_0 la tension au repos aux bornes du générateur d'impulsion et I_{sync} le courant impulsionnel de synchronisation.

Une des propriétés importantes de l'oscillateur synchrone consiste en la possibilité de se synchroniser sur une sous harmonique de sa fréquence d'oscillation f_0, c'est-à-dire lorsque la fréquence du signal de synchronisation représente une sous harmonique de f_0.

Le générateur de synchronisation qui attaque l'oscillateur génère un courant I_{sync} triangulaire de rapport cyclique ξ. Le développement en série de Fourier de ce courant donne l'expression de l'amplitude I_n de l'harmonique n :

$$I_n = \frac{2I_0}{\xi\, n^2\, \pi^2}\left[1 - \cos(\xi\, n\, \pi)\right] \qquad (3.\ 10)$$

Si l'oscillateur se synchronise à la n^{ème} sous harmonique, la largeur de la plage de synchronisation est alors donnée par l'expression suivante :

$$\Delta f = 2\frac{I_n}{V_0}\sqrt{F_b^2 + F_g^2} \qquad (3.\ 11)$$

avec I_n l'amplitude de l'harmonique n de I_{sync}.

f.ii Modélisation de l'oscillateur à résistance négative

Le schéma simplifié de l'oscillateur à résistance négative modélisé est celui de la figure 3. 27. Il est attaqué par un générateur de courant différentiel.

L'application de la théorie d'Huntoon et Weiss nous donne l'expression suivante pour la fréquence d'oscillation propre f_0 :

$$f_0 = \frac{1}{2\pi}\sqrt{\frac{1 + (2\, g_l - g_m)\, r}{L(2\, C_1 + C_2)}} \qquad (3.\ 12)$$

Avec : $C_1 = C + 2\,C_{gd}$

$$C_2 = C_{gs} + C_{gb} + C_{bd} + C_{ox1} + C_{\text{int}} + C_{buf}$$

où C_{gd}, C_{gs}, C_{gb} et C_{bd} représentent respectivement les capacités grille-drain, grille-source, grille substrat et drain-substrat d'un transistor MOS ; g_m est la transconductance des transistors NMOS de la paire différentielle ; g la charge de l'oscillateur et C_{buf} la capacité d'entrée des buffers qui constituent la charge de l'oscillateur. Dans le schéma électrique équivalent de l'inductance L figure la résistance parasite r et les capacités intrinsèque C_{int} et d'oxyde C_{ox}.

La largeur de la plage de synchronisation Δf est définie par l'expression suivante :

$$\Delta f = \frac{1}{\pi}\frac{I}{V_0}\frac{1}{2C_1 + C_2}\left(1 + \left(\frac{r}{L\,\omega_0}\right)^2\right) \qquad (3.\ 13)$$

où ω_0 est la pulsation propre de l'oscillateur, I l'amplitude du courant synchronisant l'oscillateur et V_0 l'amplitude de la tension de sortie.

g) Développement des modèles d'oscillateurs synchrones

Les modèles d'oscillateurs synchrones développés se basent sur les résultats de la théorie d'Huntoon et Weiss. Ils comportent les trois parties détection des caractéristiques d'entrée, calcul des paramètres de sortie et génération du signal de sortie.

g.i Détection des caractéristiques d'entrée

L'oscillateur est attaqué par un signal d'entrée périodique de fréquence f_i. Cette fréquence est détectée dans la première partie du modèle en insérant le processus de détection de fréquence décrit par la figure 2.16 du chapitre précédent.

g.ii Calcul des paramètres de sortie

Dans cette deuxième partie du modèle, plusieurs processus sont développés dans le but de calculer la fréquence propre de l'oscillateur, de déterminer l'harmonique de synchronisation, de calculer la plage de synchronisation et de tester finalement si l'oscillateur est ou non synchronisé. Ceci permettra de déterminer la fréquence de sortie de l'oscillateur.

Calcul de la fréquence propre f_0

La fréquence propre f_0 est calculée à partir de son expression définie par la théorie d'Huntoon et Weiss et donnée par l'équation (3. 5) ou (3. 12) selon la nature de l'oscillateur (COLPITTS ou à résistance négative). L'inductance L et les différentes capacités et transconductances sont considérées comme paramètres génériques.

```
-- oscillateur Colpitts --
-- calcul de la pulsation a vide --
wo == sqrt(((gm+gpi)*(L*gc/(c1+c2))+1.0)*(1.0/L)*(1.0/(cu+((c1*c2)/(c1+c2)))));
fo == 0.5*wo/MATH_PI;
```

figure 3. 29 : calcul de la fréquence propre f_0 de l'oscillateur COLPITTS

Calcul de l'harmonique de synchronisation

En appliquant à l'entrée de l'oscillateur un signal dont la fréquence f_i représente une sous-harmonique de sa fréquence propre f_0, l'oscillateur peut se synchroniser en donnant à la sortie un signal dont la fréquence est égale à la fréquence f_i multipliée par le numéro de sous harmonique.

Après avoir détecté f_i et calculé f_0, on peut en déduire la sous harmonique de synchronisation '*n*' en calculant la partie entière du rapport f_0/f_i. Le processus *sub_harmonic* développé est présenté par la figure 3. 30. Il est activé à chaque front montant du signal d'entrée ; donc à chaque calcul de la fréquence f_i. Ceci permettra de changer de valeur de '*n*' à chaque variation de f_i.

```
generic (seuil : real := 0.0);
quantity vin across iin through inp to inm;
quantity f0 : real;
signal y : real := 1.0;
signal n : real := 1.0;
signal periode : real;
sub_harmonic : process
begin
wait until vin'above(seuil) = true;
                -- calcul du rapport f0/fi --
```

```
y <= periode*f0;
if y <= 0.5 then n <= 1.0 ;
    -- calcul de la partie entière du rapport f0/fi --
else n <= round(y) ;
end if ;
end process sub_harmonic;
```

figure 3. 30 : calcul de la sous harmonique 'n'

Calcul de la n^{ème} harmonique du courant de synchronisation

Le générateur de courant de synchronisation qui est attaqué par le signal d'entrée génère des impulsions de courant triangulaires de rapport cyclique 'a' [Bad00]. Le calcul de l'amplitude I_n de la n^{ème} harmonique de ce courant permet de déterminer la plage de synchronisation. Elle est calculée dans les modèles à partir de l'expression (3. 10). L'amplitude I_0 et le rapport cyclique 'a' sont considérés comme paramètres génériques.

```
-- courant de synchronisation triangulaire --
i*(a*n*n*MATH_PI*MATH_PI) == 2.0*io*(1.0-cos(a*n*MATH_PI));
```

figure 3. 31 : calcul de l'harmonique n du courant de synchronisation

Calcul de la plage de synchronisation

Pour calculer la plage de synchronisation, et dans le cas d'un oscillateur COLPITTS, les facteurs de compliance F_g et F_b sont d'abord calculés à partir des équations (3. 7) et (3. 8). Ils sont fonction de la fréquence propre f_0, de l'inductance L et des différentes capacités et transconductances. La largeur de la plage de synchronisation Δf est ensuite calculée à partir de l'expression (3. 11) qui est fonction de la n^{ème} harmonique du courant de synchronisation et de la tension au repos V_0 aux bornes du générateur d'impulsions. La plage de synchronisation correspond à l'intervalle $\left[f_0 - \frac{\Delta f}{2} , f_0 + \frac{\Delta f}{2} \right]$.

```
-- oscillateur Colpitts --
-- calcul des facteurs de compliance --
cg==(1.0/(8.0*MATH_PI*MATH_PI))*(1.0/fo)*(gc/(cu*(c1+c2)+c1*c2));
cb==(1.0/(4.0*MATH_PI))*(1.0/((c1+c2)*(c1+c2)))*(L*gc*(gm+gpi)/((1.0+L*gc*(g
m+gpi)/(c1+c2)))+c1*c1/(cu+c1*c2/(c1+c2)));
-- calcul de la marge de capture --
cw == sqrt(cg*cg+cb*cb);
marge == 2.0*(i/vo)*cw;
```

figure 3. 32 : calcul de la plage de synchronisation de l'oscillateur COLPITTS

Dans le cas de l'oscillateur à résistance négative, la plage de synchronisation est calculée à partir de l'expression (3. 13) :

```
-- oscillateur à résistance négative --
-- calcul de la marge de capture --
marge == (1.0/MATH_PI)*(i/vo)*(1.0/(2.0*c1+c2))*(1.0+(r/(l*wo)/(l*wo)));
```

figure 3. 33 : calcul de la plage de synchronisation de l'oscillateur à résistance négative

Détermination de la fréquence de sortie

Pour que l'oscillateur soit synchronisé, il faut que la fréquence du courant de synchronisation appartienne à l'intervalle $\left[\frac{1}{n}\left(f_0-\frac{\Delta f}{2}\right),\frac{1}{n}\left(f_0+\frac{\Delta f}{2}\right)\right]$. Dans ce cas, la fréquence de sortie f_s est égale à la fréquence d'entrée f_i multipliée par l'harmonique 'n' ($f_s = n\, f_i$). Si l'oscillateur n'est pas synchronisé, f_s est égale à la fréquence propre f_0.

Le processus développé fait le test de synchronisation et détermine par conséquent la fréquence de sortie. Il est synchronisé sur chaque front montant d'un signal *count* dont la période est égale à la moitié de la période de sortie :

```
finf * n == f0 - 0.5*marge;
fsup * n == f0 + 0.5*marge;
synchro : process
begin
wait until count'above(half_ts_noised) = true;
        if      fin > finf and fin < fsup then
                fs <= fin * n;
```

```
        else      fs <= f0;
        end if;
end process synchro;
```

figure 3.34 : calcul de la fréquence de sortie

g.iii Génération du signal de sortie

Après avoir calculé la fréquence de sortie f_s, on peut générer dans la dernière partie du modèle le signal de sortie. Il s'agit d'un signal sinusoïdal d'amplitude V_s et de valeur moyenne V_{dd}. Ces deux tensions sont considérées comme paramètres génériques :

```
-- génération du signal de sortie --
Vout == Vdd + Vs * sin(2.0*MATH_PI*fs*NOW);
```

figure 3. 35 : génération du signal de sortie

Nous allons développé dans la suite deux exemples de modèles d'oscillateurs synchrones en présentant des résultats de simulations.

II.5 Synthèse de fréquence

Les modèles développés dans les catégories précédentes nous permettent d'élaborer des modèles structurels pour des circuits de synthèse de fréquence. Notre bibliothèque contient des modèles de boucles à verrouillage de phase et de synthétiseurs de fréquence fractionnaires.

Nous présentons à titre d'exemple dans la figure 3. 36 un modèle d'un synthétiseur de fréquence fractionnaire faisant appel aux modèles de comparateurs phase-fréquence, de pompe de charge, de filtre passe bas, de VCO et de diviseur fractionnaire. Les performances du modèle seront présentées dans le paragraphe suivant.

```
entity fractional_synthesiser is
    generic (Vhigh : real := 1.0;
    Vlow : real := 0.0;
    trise : real := 1.0e-12;
    tfall : real := 1.0e-12;
    quartz_divisor : real := 6.0;
    quartz_cyclic_ratio : real := 0.5;
    vco_gain : real := 5.0e7;
```

```
        T : real := 20.0;
        K : real := 10.0;
        C_1 : real := 15.0e-12;
        C_2 : real := 0.15e-9;
        R_2 : real := 33.6e3;
        fractional_divisor : real := 19.0;
        fractional_cyclic_ratio : real := 0.2;
        current_amplitude : real := 100.0e-6;
        delay : time := 0fs;
        vco_period : time := 3.08ns;
        input_period : time := 10ns);
        port ( terminal Tquartz, Tvco : electrical);
end entity fractional_synthesiser;

architecture structural of PLL is
        terminal Tdivider : electrical;
        terminal Treference : electrical;
        terminal Tup : electrical;
        terminal Tdown : electrical;
        terminal Tpump : electrical;
        terminal Tloop_filter : electrical;
begin
quartz_divider : entity work.frequency_divider(behavior)
                 generic map (Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall,
                 divisor=>quartz_divisor, cyclic_ratio=>quartz_cyclic_ratio,
                 delay=>delay, initial_period=>quartz_divisor*input_period)
                 port map (Tinput=>Tquartz, Toutput=>Treference);

fractional_divider : entity work.fractional_frequency_divider(behavior)
                     generic map (T=>T, divisor_N=>fractional_divisor, K=>K,
                     Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall,
                     cyclic_ratio=>fractional_cyclic_ratio,
                     delay=>delay,initial_period=>fractional_divisor*vco_period)
                     port map (Tinput_1=>Treference, Tinput_2=>Tvco, Toutput=>Tdivider);

detector :  entity work.phase_frequency_detector(behavior)
            generiv map (Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall)
            port map (Tinput_1=>Treference,Tinput_2=>Tdivider,Toutput_up=>Tup,
            Toutput_down=>Tdown);
```

```
charge :    entity work.charge_pump(behavior)
            generic map (Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall,
            current_amplitude=>current_amplitude)
            port map (Tinput_up=>Tup, Tinput_down=>Tdown, Toutput=>Tpump);

filter :    entity work.loop_filter(behavior)
            generic map (C_1=>C_1 , C_2=>C_2 , R_2=>R_2 )
            port map (Tinput=>Tpump, Toutput=>Tloop_filter);

vco :       entity work.VCO(behavior)
            generic map (Vhigh=>Vhigh, Vlow=>Vlow, trise=>trise, tfall=>tfall,
            cyclic_ratio=>fractional_cyclic_ratio, vco_gain=>vco_gain,
            delay=>delay,   vco_period=>vco_period)
            port map (Tinput=>Tloop_filter, Toutput=>Tvco);

end architecture structural;
```

figure 3. 36 : modèle structurel d'un synthétiseur de fréquence fractionnaire

II.6 Comparaison des modèles à interface analogique ou numérique

Tout au long de la partie précédente, nous avons exposé pour chaque catégorie des modèles à interface analogique. Nous avons aussi développé, comme déjà dit, des modèles à interfaces numériques et ceci dans le but de réduire le temps de simulation comportementale.

Le tableau 3. 2 compare les temps de simulations des deux types de modèles :

tableau 3. 2 : comparaison des temps de simulation

Modèles \ temps CPU	numérique	analogique	Temps d'arrêt de la simulation
Diviseur de fréquence fractionnaire	230 ms	10s 330ms	4µs
Diviseur de fréquence N/N+1	430 ms	8s 230ms	10µs
Pompe de charge	100ms	170ms	100ns
VCO	80ms	930ms	500ns

Le constat est immédiat, les simulations des modèles à interface numérique sont beaucoup plus rapides que les simulations des modèles à interface analogique dans les mêmes

conditions. Cependant, les avantages des modèles à interface analogique est l'adaptabilité des paramètres t_{rise} et t_{fall} et la compatibilité avec le simulateur SPICE.

III Performance des modèles : application à la synthèse de fréquence

Un système de radiocommunication est, comme son nom l'indique, un système de communication qui utilise les ondes hertziennes comme moyen de transmission. Les systèmes de radiocommunication sont nombreux et leurs applications très diverses : radiodiffusion, télévision, téléphonie sans fil, téléphonie portable, réseaux locaux de transmission de données numériques, etc. Généralement, ces systèmes s'appuient sur le schéma suivant :

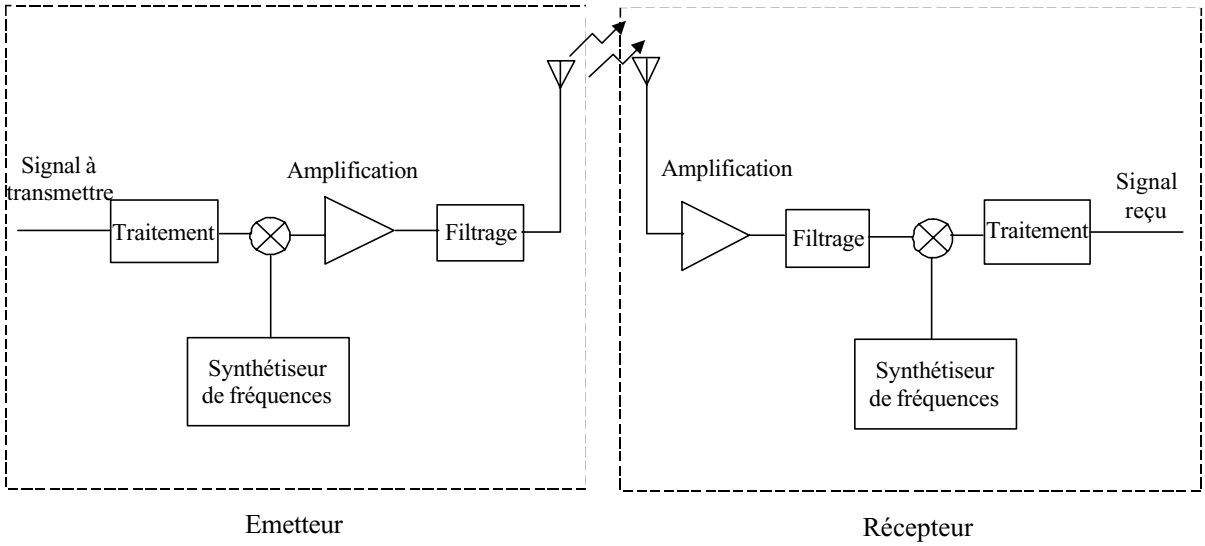

figure 3. 37 : système de radiocommunication

L'équipe de conception du laboratoire IXL s'intéresse notamment aux systèmes de synthèse de fréquence en développant des oscillateurs synchrones et des synthétiseurs de fréquence fractionnaires. Pour valider les modèles développés dans notre bibliothèque RF, nous avons comparé leurs performances à des simulations au niveau transistor ou des mesures faites sur les circuits fabriqués.

III.1 Modélisation d'un oscillateur synchrone COLPITTS 2.4 GHz dédié aux réseaux locaux sans fil

Le premier exemple d'application consiste à modéliser un oscillateur synchrone dédié aux réseaux locaux de transmission de données numériques sans fil dans la bande de

fréquence 2.40-2.49 GHz (WLAN : Wireless Local Area Network). Il a été intégré dans la technologie BiCMOS AMS 0.8 μm [Bad00].

Le cœur de l'oscillateur est un oscillateur COLPITTS totalement intégré dont le schéma électrique simplifié a été présenté par la figure 3.28. Il est attaqué par un générateur d'impulsions réalisé en logique CML qui assure la synchronisation. Il s'agit d'un train d'impulsions triangulaires d'amplitude I_0, de rapport cyclique ξ et de fréquence de répétition d'environ 400 MHz puisque l'oscillateur est synchronisé sur sa sixième sous harmonique.

III.1.1 Caractéristiques de l'oscillateur COLPITTS 2.4 GHz

La figure 3. 38 présente les résultats de la simulation du schéma transistor de l'oscillateur avec le simulateur Spectre. Elle montre le courant de synchronisation et la tension de sortie :

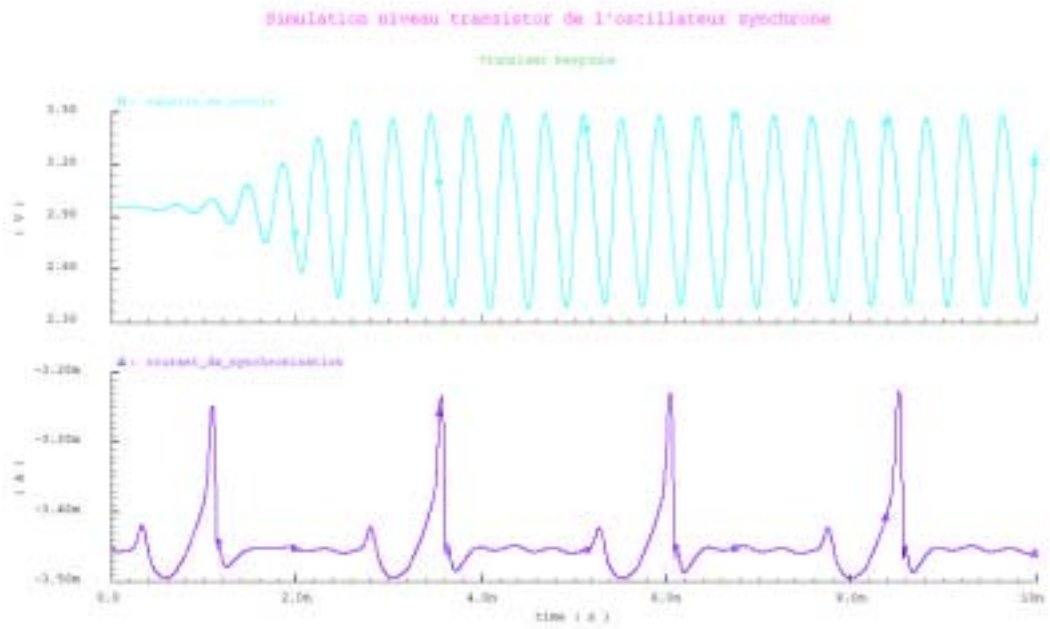

figure 3. 38 : réponse du schéma transistor de l'oscillateur

Le simulateur SpectreRF nous permet de déterminer la plage de synchronisation de l'oscillateur pour différentes valeurs de l'amplitude du courant de synchronisation. La courbe de simulation obtenue est donnée par la figure 3. 39 :

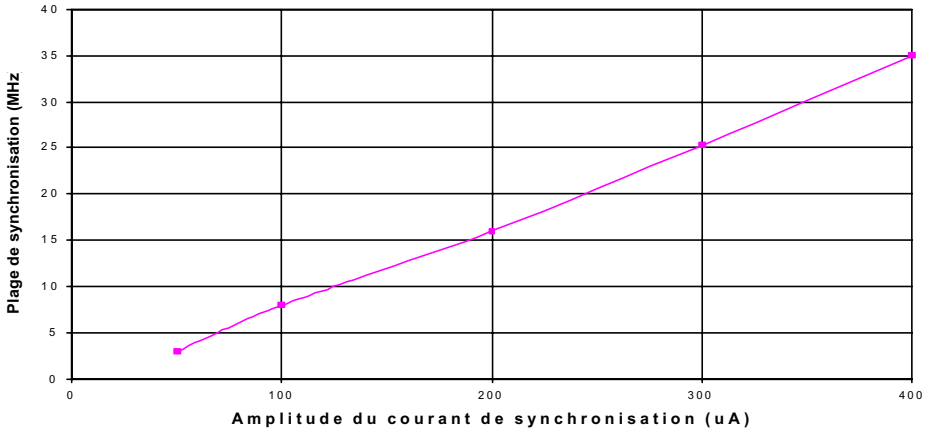

figure 3. 39 : Plage de synchronisation simulée de l'oscillateur 'poly'

La fréquence propre de l'oscillateur synchrone est égale à 2.398 GHz et sa plage de synchronisation est de 8 MHz pour une amplitude du courant de synchronisation égale à 100μA.

III.1.2 Résultats de la simulation comportementale

Les résultats des simulations niveau transistor nous ont permis de déterminer les valeurs des capacités parasites et des transconductances utilisées dans la théorie d'Huntoon et Weiss. Ils nous ont également permis d'avoir les valeurs de l'amplitude du signal aux bornes du générateur de synchronisation V_0 et celle du signal de sortie V_s, considérées comme paramètres génériques dans notre modèle d'oscillateur.

Une fois les valeurs des paramètres génériques fixées, nous avons appliqué à l'entrée du modèle de l'oscillateur un signal périodique de fréquence 398.4 MHz. Pour cette fréquence d'entrée, l'oscillateur n'est pas synchronisé. Nous obtenons alors en sortie un signal de fréquence f_0 égale à 2.398 GHz. Pour une amplitude du courant du signal de synchronisation égale à 100 μA, la plage de capture Δf calculée est de 8 MHz. Ces deux valeurs pour f_0 et Δf correspondent bien aux résultats de la simulation niveau transistor.

En faisant varier l'amplitude du signal de synchronisation, nous avons tracé la caractéristique de la plage de synchronisation, présentée par la figure 3. 40. Elle est comparée à celle trouvée avec le simulateur spectreRF. Pour les faibles courants de synchronisation, les résultats obtenus par les deux simulations coïncident. Pour les forts courants, nous observons un écart croissant qui s'explique par les deux points suivants :

- Dans le schéma équivalent de l'oscillateur qui nous a permis de développer l'expression de la plage de synchronisation, nous n'avons pas tenu compte de tous les éléments parasites présents dans le circuit ;

- Pour les forts courants de synchronisation, il faut modifier l'expression de la condition d'oscillation donnée par la théorie d'Huntoon et Weiss [Bad00], ce qui modifie l'expression de la plage de synchronisation.

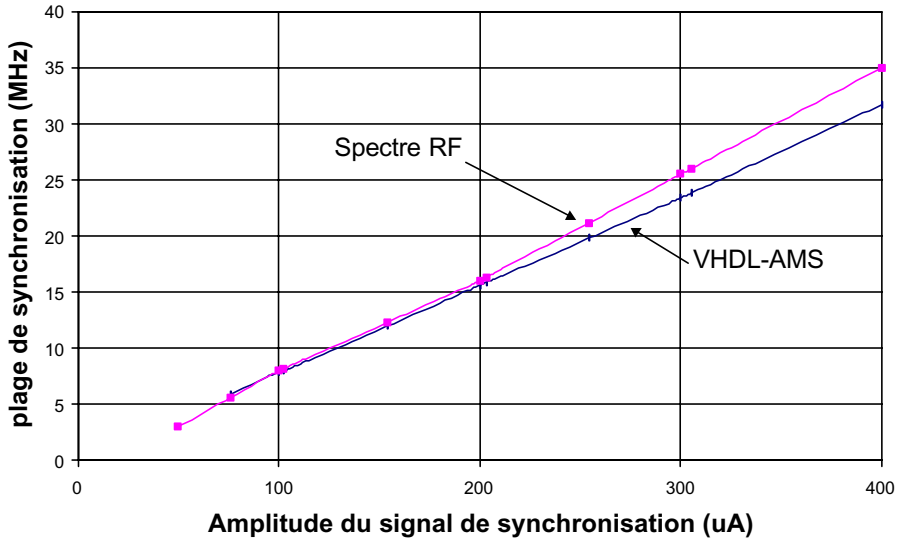

figure 3. 40 : comparaison des plages de synchronisation simulées

Les résultats obtenus valident notre modèle d'oscillateur synchrone COLPITTS 2.4GHz. Pour une simulation allant jusqu'à 10ns correspondant à 24 périodes du signal de sortie, le temps de simulation du schéma transistor simplifié de l'oscillateur avec SpectreRF est de l'ordre de 20s ; celui de la simulation du modèle avec ADVance_MS est seulement de 0.41s. Ceci correspond à un gain en simulation comportementale de l'ordre de 50.

III.2 Modélisation d'un oscillateur synchrone 5.2 GHz dédié aux applications de type HIPERLAN

Un deuxième modèle d'oscillateur synchrone a été développé pour un deuxième prototype fabriqué. Il s'agit d'un oscillateur synchrone 5.2 GHz intégré dans la technologie HCMOS7 (technologie CMOS 0.25 µm) de STMicroelectronics, dédié aux applications de type HIPERLAN dont la fréquence de travail entre dans la gamme 5.15-5.25 GHz [Bad00].

Pour des raisons d'optimisation de la plage de synchronisation, l'oscillateur conçu est un oscillateur à résistance négative dont le schéma électrique de principe a été présenté par la figure 3. 27. Il est attaqué par un générateur de synchronisation différentiel.

III.2.1 Caractéristiques de l'oscillateur 5.2 GHz

Le tableau 3. 3 résume les principaux résultats issus des simulations niveau transistor de l'oscillateur synchrone. Il est synchronisé sur sa sixième sous harmonique. La plage de synchronisation a été relevée pour une amplitude de courant de synchronisation égale à 420μA.

tableau 3. 3 : principaux résultats de simulation

Fréquence des oscillations libres	5.7 GHz
Amplitude des oscillations	1.37 V
Plage de synchronisation simulée	160 MHz (i = 420 μA)

III.2.2 Résultats de la simulation comportementale

Les valeurs des capacités parasites et des transconductances utilisées dans la théorie d'Huntoon et Weiss pour déterminer la fréquence propre de l'oscillateur et sa plage de capture sont déterminées par des simulations niveau transistor. Les résultats de la simulation comportementale nous ont donné une fréquence propre f_0 égale à 5.7 GHz et une plage de capture de 156 MHz pour un courant de synchronisation d'amplitude 420 μA à sa sixième sous harmonique. Ces résultats vérifient ceux issus des simulations niveau transistor.

Ces oscillateurs ont également fait l'objet d'une étude et d'une modélisation de bruit de phase qui seront présentées dans le chapitre 4.

III.3 Modélisation d'un synthétiseur de fréquence fractionnaire conforme à la norme UMTS

Les synthétiseurs de fréquence sont des circuits qui permettent de synthétiser une bande de fréquences à partir d'une fréquence de référence appliquée à l'entrée. Ils sont généralement répartis en deux grandes familles : les synthétiseurs de fréquence directs et indirects. Nous nous sommes intéressé aux synthétiseurs de fréquence indirects qui utilisent

généralement des boucles à verrouillage de phase pour générer l'oscillateur local. En particulier, nous allons étudier les synthétiseurs fractionnaires.

Un synthétiseur de fréquence est caractérisé par sa plage d'accord, son pas de synthèse, son temps d'établissement, les raies parasites présentes dans le spectre du signal de sortie et ses caractéristiques en terme de bruit de phase. On pourra ajouter également les critères de consommation et d'encombrement :

- **Plage d'accord** : La plage d'accord est la plage de fréquence qui doit être synthétisée. Elle est fonction de l'application radio fréquence envisagée.
- **Pas de synthèse** : Le pas de synthèse est la différence de fréquence entre deux canaux adjacents. Elle est généralement égale à la largeur du canal de l'application considérée.
- **Temps d'établissement** : C'est le temps que met le synthétiseur pour passer d'un état stable à un autre. Sa définition dépend de l'application. Il peut être relié à une déviation maximale de la fréquence ou de la phase.
- **Spurious** : Les spurious sont des raies parasites autres que les harmoniques de la fréquence synthétisée et qui sont inhérentes au processus de synthèse.
- **Bruit de phase** : Les différentes sources de bruit présentes dans un oscillateur font que celui-ci a une fréquence qui est fonction du temps et son expression temporelle est de la forme :

$$V(t) = f\left(2\pi f_0 t + \phi(t)\right) \qquad (3.\ 14)$$

$\phi(t)$ étant l'excès de phase dû aux sources de bruit.

La représentation spectrale de V(t) fait apparaître deux lobes de part et d'autre de la raie centrale f_0 dus au terme $\phi(t)$.

III.3.1 Principe général de fonctionnement

h) Synthétiseur de fréquence classique

La figure 3. 41 présente l'architecture classique 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. L'architecture du synthétiseur de fréquence est composée des éléments suivants : un comparateur phase-

fréquence suivi d'une pompe de charge, un filtre de boucle, un oscillateur contrôlé en tension et un diviseur de fréquence. La sélection du canal à synthétiser se fait en agissant sur le rapport de division N, comme le montre la figure 3. 41.

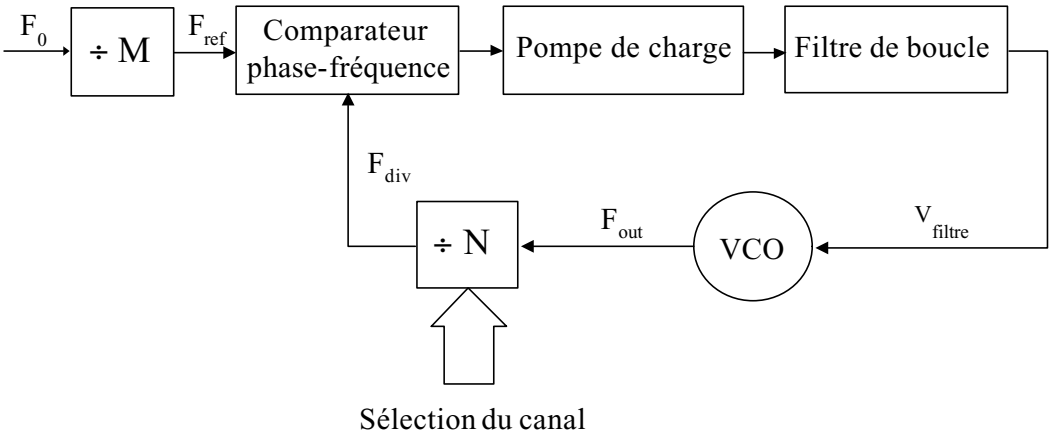

figure 3. 41 : Synthétiseur de fréquence à base de PLL

Lorsque les deux signaux $V(F_{div})$ et $V(F_{ref})$ ne sont pas en phase, le comparateur de phase génère une tension d'erreur. Cette tension est filtrée par le filtre de boucle et sa valeur moyenne pilote directement le VCO à travers la contre-réaction. Lorsque la boucle est verrouillée, la tension à la sortie du filtre est constante ainsi que la différence entre les phases $\varphi_{div}(t)$ et $\varphi_{ref}(t)$. Ceci implique l'égalité entre les deux fréquences F_{div} et F_{ref} puisque la fréquence d'un signal est la dérivée de sa phase par rapport au temps. Cette dernière relation peut alors s'écrire :

$$F_{out} = \frac{N}{M} F_0 \qquad (3.\ 15)$$

Pour réaliser un synthétiseur de fréquence, deux possibilités sont offertes :

- Soit faire varier la fréquence de référence. La plage de variation de la fréquence de sortie est alors égale à la plage de variation de F_{ref} multipliée par N.
- Soit utiliser une fréquence de référence fixe, généralement générée à partir d'un oscillateur à quartz, et faire varier le rapport de division N en utilisant un compteur programmable dans la boucle de retour.

Les synthétiseurs de fréquence fractionnaires utilisent la deuxième solution pour générer toute une plage de fréquence.

i) Synthétiseur de fréquence fractionnaire

Le pas de synthèse d'un synthétiseur classique est égal à la fréquence de référence. En effet, si on passe d'un rapport de division N à un rapport N+1 on aura :

$$F_{out} = \frac{N+1}{M} F_0 = \frac{N}{M} F_0 + \frac{1}{M} F_0 = \frac{N}{M} F_0 + F_{ref} \qquad (3.\ 16)$$

Pour diminuer le pas de synthèse, il faut donc diminuer la fréquence de référence F_{ref}. Cependant, ceci augmentera le bruit de phase du synthétiseur de fréquence. En effet, en diminuant F_{ref}, on diminue F_{div} et donc on augmente le rapport de division N. La période du signal de sortie du VCO est dans ce cas multipliée par un entier plus grand et donc le bruit de phase à la sortie du diviseur sera plus élevé.

Pour maintenir le pas de synthèse réduit, on peut le forcer à être une fraction de F_{ref}. Le diviseur par N de la boucle de retour doit alors être capable de manipuler des nombres réels et non pas seulement des entiers : c'est le principe de la synthèse fractionnaire.

La figure 3. 42 illustre le principe de fonctionnement d'un synthétiseur fractionnaire. Le diviseur de fréquence par N de la figure 3. 41 est remplacé par un diviseur N/N+1 et un accumulateur. Le comparateur de phase est remplacé par un comparateur phase fréquence suivi d'une pompe de charge.

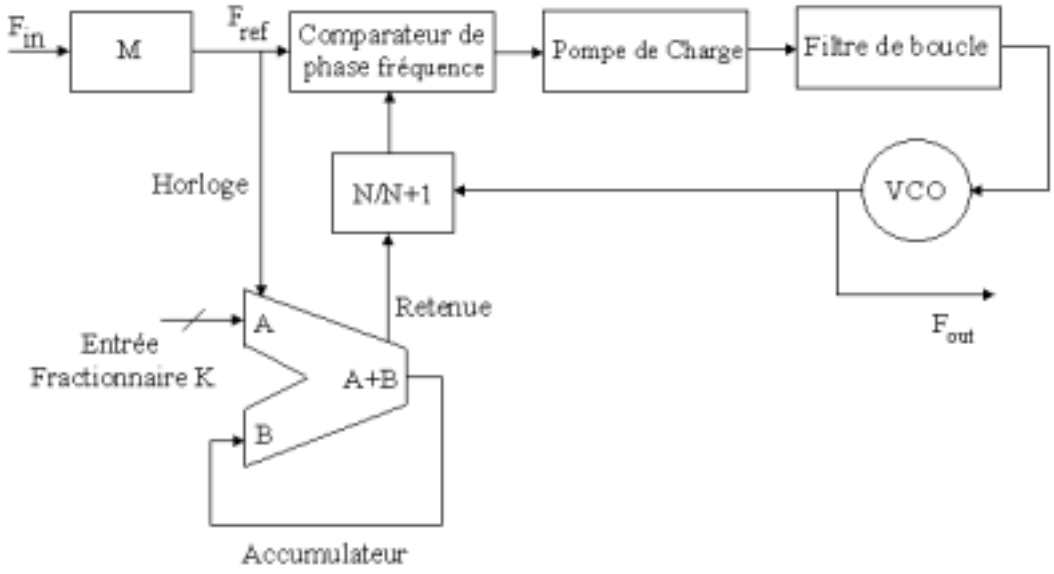

figure 3. 42: synthétiseur fractionnaire

III.3.2 Détermination des paramètres génériques du synthétiseur fractionnaire

Le schéma équivalent du synthétiseur fractionnaire modélisé est celui de la figure 3. 42. La sortie du VCO pilote un oscillateur synchrone qui se synchronise sur la sixième sous harmonique de son signal d'entrée.

Le modèle développé pour le synthétiseur est celui présenté par la figure 3. 36. Il répond à la norme UMTS dont les caractéristiques se résument en les points suivants :

- 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 (24 dBm)

La fréquence à la sortie du VCO varie donc de 320 MHz à 330 MHz, ce qui correspond à la bande d'émission / réception divisée par 6.

Le tableau 3. 4 résume les différents paramètres génériques du synthétiseur fractionnaire :

tableau 3. 4 : principaux paramètres génériques du synthétiseur fractionnaire

Sous circuits	**Paramètres**	**Valeurs**
Quartz	Fréquence de référence : $\mathbf{F_{ref}}$	100 kHz
Diviseur de fréquence M	Rapport de division : **M**	6
Diviseur de fréquence N/N+1	Rapport de division : **N**	
Pompe de charge	Courant de charge : $\mathbf{I_{pompe}}$	100 μA
Filtre de boucle	Résistance : **R** Capacités : **C1, C2**	
VCO	Sensibilité : $\mathbf{K_{VCO}}$ Fréquence propre : $\mathbf{f_0}$	50 MHz/V 325 MHz
Accumulateur	Dimension de l'accumulateur : **T**	

Certains paramètres ont été fixés selon les spécifications des circuits transistors utilisés pour réaliser le synthétiseur de fréquence. A partir de ces valeurs, nous avons déterminé la

dimension T de l'accumulateur, le rapport de division N et les paramètres du filtre R, C_1 et C_2. Le calcul correspondant est donné en annexe.

III.3.3 Simulation comportementale du synthétiseur fractionnaire

La figure 3. 43 récapitule les différentes valeurs des paramètres de notre synthétiseur fractionnaire :

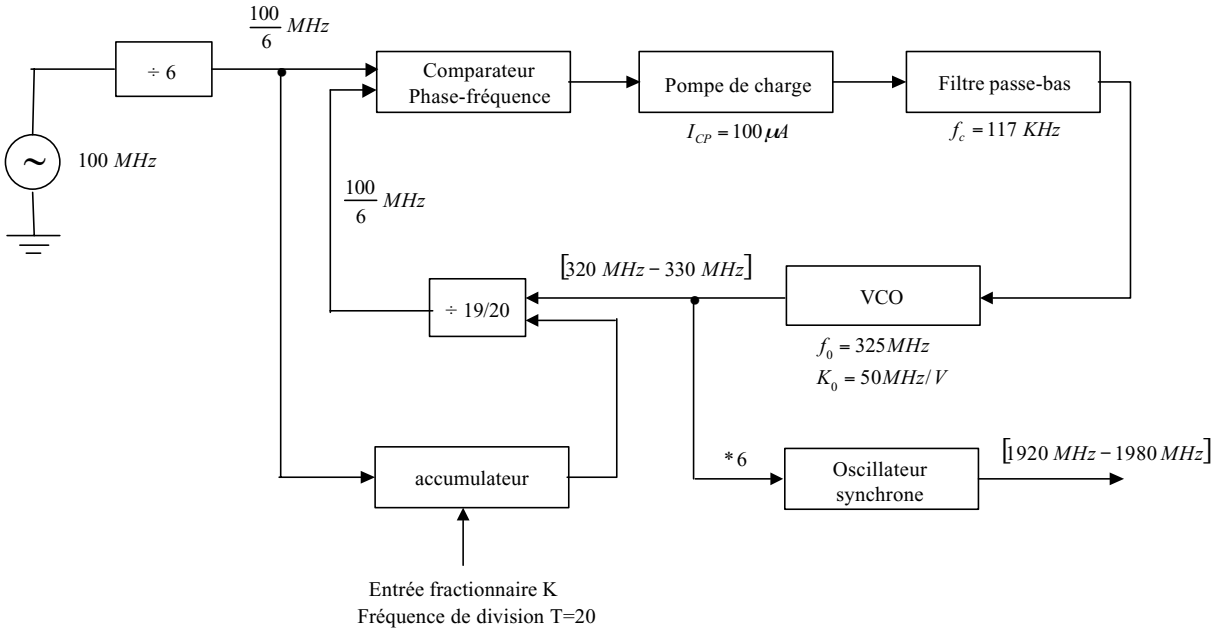

figure 3. 43 : caractéristiques du synthétiseur fractionnaire

La figure 3. 44 illustre les réponses du modèle de synthétiseur fractionnaire suite à un changement de la valeur du paramètre K (4, 10, 16), ce qui induit un changement dans la valeur du rapport de division moyen N_{FRAC} (19.2, 19.5, 19.80).

La première courbe montre la variation du rapport de division N_{FRAC} ; la deuxième illustre le courant à la sortie de la pompe de charge (ce qui correspond à la sortie du filtre de boucle) et la troisième courbe montre la variation de la période de sortie du VCO suite au changement du rapport de division. Les résultats obtenus prouvent le bon fonctionnement du synthétiseur et donc confirme la bonne modélisation de chaque bloc.

La figure 3. 45 montre les résultats de simulation à partir des modèles à interface numérique du synthétiseur de fréquence. On remarque le bon accord avec les résultats de simulation à partir des modèles à interface analogique de la figure 3. 44.

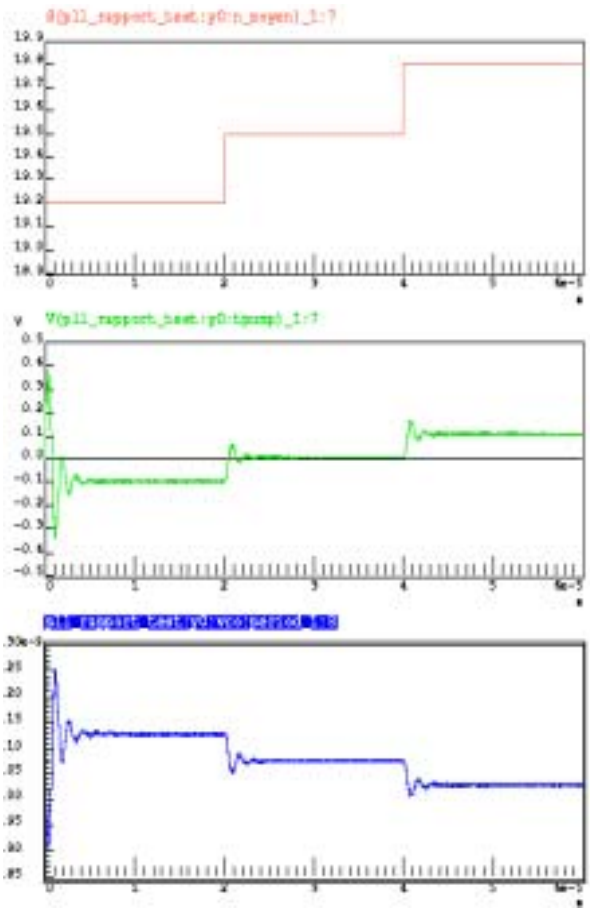

figure 3. 44 : Simulation du modèle analogique du synthétiseur de fréquence

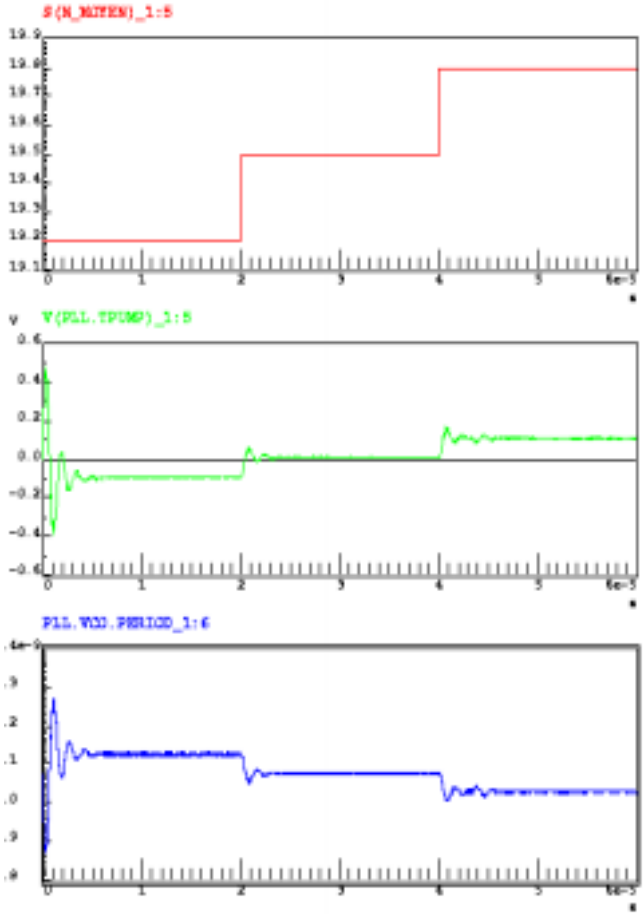

figure 3. 45 : Simulation du modèle numérique du synthétiseur de fréquence

Le temps de simulation du modèle analogique est de 13min ; celui du modèle numérique est de 12s pour un temps d'arrêt de 60µs pour la simulation. Là encore les temps de simulation diffèrent suivant le type d'interface utilisé, néanmoins ils restent sans commune mesure avec ceux des simulations au niveau transistor sous CADENCE qui peuvent atteindre des journées entières.

L'utilisation de la modélisation comportementale se justifie pour ce genre d'applications. 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.

IV Conclusion

Nous avons présenté dans ce chapitre une bibliothèque de modèles comportementaux développée pour des applications de synthèse de fréquence. L'approche adoptée pour développer ces modèles était dans la plut part des cas une approche fonctionnelle.

La bibliothèque a été utilisée pour développer des modèles comportementaux pour trois applications : un oscillateur synchrone COLPITTS 2.4 GHz dédié aux réseaux locaux de transmission de données numériques sans fil ; un oscillateur synchrone à résistance négative 5.2 GHz dédié aux applications de type HIPERLAN et un synthétiseur de fréquence fractionnaire répondant à la norme UMTS.

Les comparaisons faites entre les performances des modèles et les simulations au niveau transistor avec CADENCE ou les mesures sur les circuits fabriqués ont permis de valider les modèles.

Pour les systèmes de radiocommunication, il est important d'étudier le bruit de phase qui introduit des erreurs sur le spectre des signaux de sortie. Nous allons étudier dans le chapitre suivant ce phénomène pour tenir compte de son effet dans nos modèles d'oscillateurs.

Chapitre 4

Bruit de phase dans les oscillateurs : Théorie et modélisation

I Introduction

Pour concevoir des systèmes Radio Fréquences fiables, il est important d'optimiser le bruit de phase qui les caractérise. Cette optimisation peut se réaliser à deux niveaux : le haut niveau en faisant un choix optimisé de l'architecture du système RF et le bas niveau en minimisant le bruit qu'introduisent les différents composants électroniques.

L'optimisation du choix de l'architecture peut être faite par simulation comportementale. Pour optimiser le bruit de phase à ce niveau, il faut que les modèles développés tiennent comptent de son effet. C'est dans ce but que nous avons développé une méthode de modélisation comportementale du bruit de phase.

Dans un circuit de synthèse de fréquence, l'oscillateur local introduit un bruit de phase conséquent. Nous nous intéressons dans ce chapitre à la modélisation du bruit de phase dans ce type de circuit. Le bruit étant un signal aléatoire, nous commençons alors ce chapitre par donner un rappel sur ce type de signaux. Nous étudions ensuite le bruit de phase dans les oscillateurs et nous détaillons notre approche de modélisation pour tenir compte de son effet dans les modèles d'oscillateurs. Finalement, et comme application, nous présentons les résultats de la modélisation comportementale du bruit de phase dans les oscillateurs synchrones.

II Généralités sur les signaux aléatoires

Les signaux peuvent être classifiés en deux catégories : les signaux déterministes et les signaux aléatoires. Avant d'introduire les outils mathématiques d'investigation des signaux aléatoires, nous revenons brièvement sur le cas des signaux déterministes [Jel96].

II.1 Rappels sur les signaux déterministes

Ils comprennent les signaux reproductibles dont la forme peut être reproduite avec une approximation suffisamment bonne pour qu'on puisse la considérer comme identique d'une expérience à l'autre [Cha90]. Ils sont classés en deux catégories : les signaux transitoires et les signaux permanents.

II.1.1 Les signaux transitoires

Ce sont des signaux qui ont une durée qui n'excède pas la durée de l'observation ou de l'expérience. C'est le cas par exemple de transitoires exponentiels :

$$x(t) = e^{-\alpha} \cos(2\pi f t + \theta) \qquad t > 0 \tag{4. 1}$$

II.1.2 Les signaux permanents

Ils ont une durée supposée infinie, soit par ignorance de leur durée, soit par choix d'un modèle mathématique. C'est par exemple le cas des signaux modulés en fréquence, tel que la sinusoïde :

$$x(t) = A \sin(\phi(t)) \tag{4. 2}$$

On définit :

• Les signaux à énergie finie, tels que :

$$E = \int_{\Re} |x(t)|^2 dt < \infty \tag{4. 3}$$

Cette classe recouvre les signaux physiques transitoires.

• Les signaux à puissance moyenne finie tels que:

$$P = \lim_{T \to \infty} \frac{1}{T} \int_{-T/2}^{+T/2} |x(t)|^2 dt < \infty \tag{4. 4}$$

Cette classe recouvre les signaux physiques permanents.

Pour les signaux à temps discret, les intégrales sont à remplacer par des sommes simples.

II.1.3 Transformation de Fourier

Pour les signaux déterministes d'allure complexe, l'information pertinente n'est pas contenue dans la forme d'onde x(t) elle-même. De très nombreuses transformations sont disponibles qui produisent chacune une représentation différente du signal. La plupart de ces

représentations reposent sur des transformations intégrales ou leurs contreparties discrètes. Parmi les représentations classiques, on trouve la **Transformée de Fourier (TF)** donnée pour un signal x(t) par :

$$X(f) = TF\left[x(t)\right] = \int_{\Re} x(t)\, e^{-i2\pi ft} dt \tag{4. 5}$$

En corollaire de cette propriété intervient le théorème de convolution :

$$TF\left\{x(t) * y(t)\right\} = X(f).Y(f) \tag{4. 6}$$

Où l'opérateur de convolution * s'écrit :

$$x(t) * y(t) = \int_{\Re} x(u)y(t-u)dt \tag{4. 7}$$

Le produit de convolution décrit la transformation entrée-sortie des systèmes linéaires invariants à temps continu.

La transformée inverse de la transformation de Fourier est donnée par :

$$x(t) = TFI\left[X(f)\right] = \int_{\Re} X(f)\, e^{i2\pi ft} df \tag{4. 8}$$

Notons une relation simple qui a une grande importance pratique :

$$TFI\left[\left|X(f)\right|^2\right] = \int_{\Re} x(u)\, x^*(u-t)\, du \tag{4. 9}$$

$\left|X(f)\right|^2$ représente la **densité spectrale d'énergie**.

$\int_{\Re} x(u)\, x^*(u-t)\, du$ représente la **fonction d'auto-corrélation temporelle**.

II.2 Les signaux aléatoires

Les signaux aléatoires sont ceux qui admettent une représentation probabiliste. On admet que le signal observé est membre d'une « famille » de signaux, définie par ses propriétés statistiques (densité de probabilité, fonction de répartition, moments, etc…) [MaE97].

II.2.1 Densité de probabilité

a) Variable aléatoire discrète

Une variable aléatoire discrète X est une variable aléatoire à valeurs dans un ensemble dénombrable de valeurs réelles $\{a_1, a_2, \ldots, a_k, \ldots\}$. Sa **densité de probabilité** p_k est définie comme suit :

$$prob(X = a_k) = p_k \quad avec \quad p_k \geq 0 \quad \forall k \qquad (4.\ 10)$$
$$avec \qquad \sum_k p_k = 1$$

On définit la **fonction de répartition** $F_x(a)$ comme étant égale à $Prob(X \in]-\infty, a])$. C'est une fonction croissante qui tend vers 0 quand a tend vers $-\infty$ et qui tend vers 1 quand a tend vers $+\infty$.

La fonction de répartition d'une variable aléatoire discrète est une fonction en escalier présentant un nombre fini de discontinuités de première espèce. Les marches ont une hauteur égale à l'amplitude des probabilités :

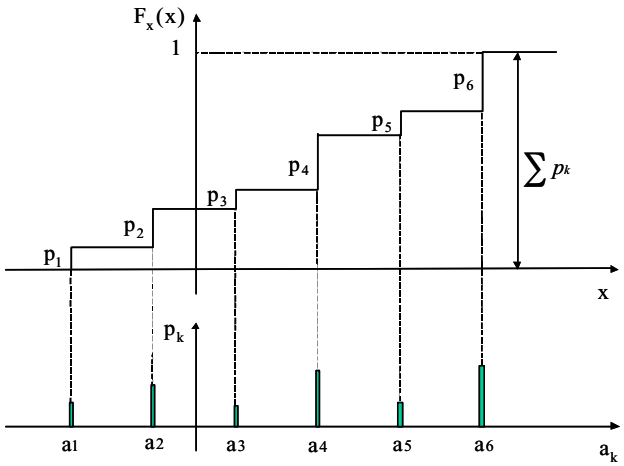

figure 4. 1 : fonction de répartition

b) Variable aléatoire continue

Une variable aléatoire continue X est une variable à valeurs dans $\Re$. Sa **densité de probabilité** est définie comme suit :

$$prob\,(X \in]a,b]\,) = \int_a^b p_x(z)dz \qquad (4.\ 11)$$
$$avec \qquad p_x(z) \geq 0 \quad \forall z$$
$$et \qquad \int_{\Re} p_x(z)\,dz = 1$$

Par exemple, la densité de probabilité d'une variable aléatoire Gaussienne est donnée par :

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\left(e^{-\frac{x^2}{2\sigma^2}} \right) \qquad (4.\ 12)$$

Cette densité de probabilité est représentée par la figure 4. 2 :

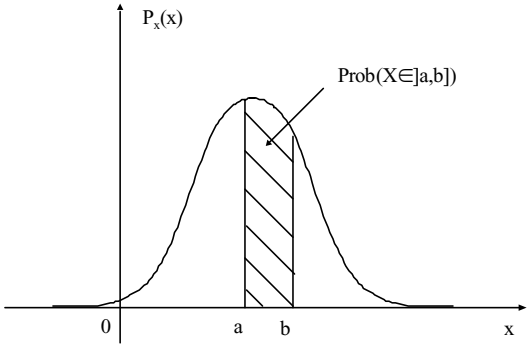

figure 4. 2 : densité de probabilité Gaussienne

II.2.2 Moment d'une variable aléatoire

L'espérance mathématique de la variable aléatoire X, que l'on note E(X), est définie (lorsqu'elle existe) de la façon suivante:

- Pour une variable aléatoire discrète :

$$E(X) = \sum_k a_k p_k \qquad (4.\ 13)$$

où {a1, a2, …, ak, …} désigne l'ensemble des valeurs possibles.

- Pour une variable aléatoire continue :

$$E(X) = \int_{\Re} z\ p_x(z)\,dz \qquad (4.\ 14)$$

On appelle **moment d'ordre n** l'espérance mathématique de la variable aléatoire X^n.

Pour une variable aléatoire discrète il vient :

$$E(X^n) = \sum_k a_k^n \; p_k \tag{4. 15}$$

Pour une variable aléatoire continue il vient :

$$E(X^n) = \int_{\Re} z^n p_x(z)\, dz \tag{4. 16}$$

La valeur pour n=1 s'appelle la **moyenne**. Quand la valeur moyenne est nulle, on dit que la variable aléatoire est **centrée**.

La valeur pour n=2 s'appelle le **moment du second ordre**. Il s'interprète en théorie du signal comme une puissance.

On appelle **variance** la quantité :

$$\sigma^2 = E((X - E(X))^2) \tag{4. 17}$$

En développant le carré et en utilisant l'équation ((4. 14) on obtient :

$$\sigma^2 = E(X^2) - E^2(X) \tag{4. 18}$$

L'écart type σ est défini comme étant la racine carrée de la variance.

II.2.3 Stationnarité et ergodicité

Une variable aléatoire X est **stationnaire** si toutes ses propriétés statistiques sont invariantes dans le temps, donc indépendantes du choix de l'origine des temps. Par exemple, si les densités de probabilités ont cette propriété, on aura :

$$p_x(z,t) = p_x(z) \quad ou \quad p_x(z,y,t_1,t_2) = p_x(z,y,t_1 - t_2) \tag{4. 19}$$

Une variable aléatoire X est **ergodique** si les moyennes statistiques (moments) sont égales aux moyennes temporelles correspondantes. On aura donc :

$$E(x) = \int_{-\infty}^{+\infty} x\; p(x)\, dx = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t)\, dt \tag{4. 20}$$

$$et \quad E(x^2) = \int_{-\infty}^{+\infty} x^2 \; p(x)\, dx = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x^2(t)\, dt \tag{4. 21}$$

II.2.4 Fonction de corrélation et densité spectrale de puissance

Pour les signaux aléatoires, on ne dispose pas d'une écriture mathématique décrivant leur répartition temporelle x(t) et donc pas non plus de l'expression de leur transformé de Fourier donnant leur répartition dans le domaine des fréquences. Pour les décrire, on définit les fonctions de corrélation et les densités spectrales de puissance.

a) Fonction de corrélation

La fonction de corrélation $C_{xx}(\tau)$ d'un signal x(t) est définie comme suit :

$$C_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t)\; x^*(t-\tau)\, dt \tag{4. 22}$$

La fonction de corrélation a les propriétés suivantes :

- Symétrie hermitienne, qui ce réduit à la symétrie simple pour les signaux réels :

$$C_{xx}(\tau) = C_{xx}{}^*(-\tau) \tag{4. 23}$$

- Bornitude :

$$\left|C_{xx}(\tau)\right| \le C_{xx}(0) \tag{4. 24}$$

Elle admet donc un maximum à l'origine. On montre que s'il est atteint pour une autre valeur de τ que 0, il est atteint périodiquement.

- Continuité : Si une fonction de corrélation est continue à l'origine, elle est continue partout. C'est le cas pour tous les signaux physiques.

Dans le cas où il n'existerait aucune relation entre les valeurs du signal x(t) aux différents instants t, la fonction de corrélation $C_{xx}(\tau)$ sera nulle partout sauf en τ=0. La quantité $C_{xx}(0)$ représente la **valeur quadratique moyenne** du signal :

$$C_{xx}(0) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x^2(t)\, dt \qquad (4.\ 25)$$

Un bruit électronique dont la fonction de corrélation $C_{xx}(\tau)$ est nulle partout sauf en τ=0 est appelé **bruit blanc**. En générale, les sources de bruit présentent des fonctions de corrélation avec l'allure suivante :

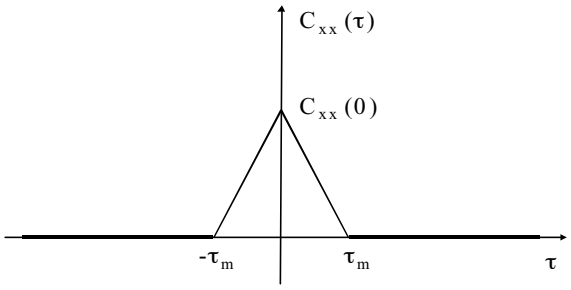

figure 4. 3 : exemple d'allure de fonction de corrélation

Dans le cas des signaux périodiques où x(t+T) = x(t), on définit la fonction de corrélation suivante :

$$C_{xx}(\tau) = \frac{1}{T} \int_{\theta}^{\theta+T} x(t)x(t-\tau)dt \qquad (4.\ 26)$$

b) Densité spectrale de puissance

La fonction de corrélation $C_{xx}(\tau)$ ne nous renseigne pas directement sur la manière dont l'énergie est répartie dans le domaine des fréquences. Pour répondre à cette question, on introduit la **densité spectrale de puissance** du signal. Soit s(t) le signal obtenu après le filtrage du signal x(t) autour de la fréquence f. La puissance moyenne P_s de s(t) est donnée par [Cou00] :

$$P_s = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s^2(t)\, dt \qquad (4.\ 27)$$

La **densité spectrale de puissance X(f)** est par définition la puissance moyenne du signal filtré autour d'une fréquence, soit :

$$X(f) = \frac{P_s}{df} \qquad (4.\ 28)$$

La quantité $\int_{-\infty}^{+\infty} X(f)df$ représente la puissance moyenne P_x du signal x(t) qui est égale à la puissance moyenne exprimée dans le domaine temporel ; on obtient donc la relation importante suivante :

$$P_x = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x^2(t)\, dt = \int_{-\infty}^{+\infty} X(f)\, df \qquad (4.\ 29)$$

On montre également que la densité spectrale de puissance X(f) est la transformée de Fourier de la fonction de corrélation $C_{xx}(\tau)$ (théorème de Wiener-Kinchine) :

$$X(f) = TF\left(C_{xx}(\tau)\right) = \int_{-\infty}^{\infty} C_{xx}(\tau) e^{-j2\pi f\tau} d\tau \qquad (4.\ 30)$$

Dans le cas particulier où la fonction de corrélation $C_{xx}(\tau)$ est nulle partout sauf en τ=0 où elle vaut $C_{xx}(0)$ (cas du bruit blanc), la densité spectrale X(f) est donnée par :

$$X(f) = C_{xx}(0) \qquad (4.\ 31)$$

Si la densité spectrale X(f) est une constante quelle que soit la fréquence cela veut dire que le signal x(t) comprend toutes les fréquences avec une égale amplitude.

III Etude du bruit de phase dans les oscillateurs

III.1 Caractérisation du bruit de phase dans le domaine fréquentiel

III.1.1 Densité spectrale de puissance du bruit de phase

Le bruit introduit des erreurs sur l'amplitude et sur la phase de la tension de sortie des oscillateurs. L'erreur sur l'amplitude peut être limitée par des dispositifs de contrôle automatique de gain. En ne tenant pas compte de cette erreur, la tension de sortie bruitée d'un oscillateur peut être alors exprimée par l'équation suivante :

$$V(t) = f\left(2\pi f_0 t + \phi(t)\right) \qquad (4.\ 32)$$

Avec : f_0 la fréquence centrale, appelée fréquence de la porteuse (carrier frequency)

$\phi(t)$ l'excès de phase dû aux sources de bruit

Le bruit de phase de l'oscillateur est représenté dans l'équation (4. 32) par le processus aléatoire $\phi(t)$. On note S_ϕ la **densité spectrale de puissance** de la phase $\phi(t)$. S_ϕ n'est pas une quantité directement mesurable ; par contre S_v qui est la densité spectrale de puissance du signal V(t) se mesure à l'aide d'un analyseur de spectre. Plus exactement, on mesure une densité de puissance normalisée par rapport à la puissance du fondamental, et ce à un offset f_m de la fréquence centrale. Elle est notée $L(f_m)$, exprimée en dB_c/Hz et définie par [Haj99] :

$$\left|L(f_m)\right|_{dB_c/Hz} = 10\log\left[\frac{P(f_0 + f_m, 1Hz)}{P_0}\right] \qquad (4.\ 33)$$

Avec : $P(f_0+f_m,1Hz)$ la puissance de V(t) à un offset f_m de la porteuse f_0 et calculée dans une bande passante de 1Hz

et P_0 la puissance du fondamental de V(t)

Pour les fréquences où $\mathbf{S_\phi(f_m)}$ est faible, on démontre la relation suivante [Kun98] :

$$S_\phi(f_m) = 2L(f_m) \qquad (4.\ 34)$$

La figure 4. 4 illustre la densité spectrale de puissance du signal V(t).

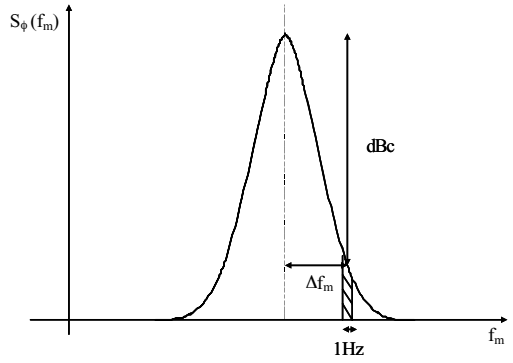

figure 4. 4 : densité spectrale de puissance du signal v(t)

III.1.2 Allure de la densité spectrale de puissance du bruit de phase

Les sources de bruit en électronique sont généralement des bruits blancs ou roses. Un bruit blanc est un signal issu d'une suite de variables aléatoires non corrélées et en général centrées ; ce qui permet de calculer toutes ses propriétés du second ordre [Dqu96]. Sa densité

spectrale de puissance est par conséquent constante sur tout l'axe des fréquences. Le bruit thermique ou le bruit de grenaille sont par exemple des bruits blancs.

Un bruit rose est un bruit dont la densité spectrale de puissance est dominante en basse fréquence. Les bruits en 1/f, $1/f^2$, $1/f^3$, etc. sont tous des bruits roses. Le bruit de Flicker présente par exemple un bruit en 1/f. Ce type de bruit ne présente plus d'effets à partir d'une fréquence f_a par rapport à la fréquence centrale f_0 du signal bruité. Cette fréquence est fonction de plusieurs paramètres intérieurs et extérieurs pour un circuit donné. Elle est difficile à calculer et est déterminée par des mesures. Typiquement, elle est égale à $10^{-5} \times f_0$ [Wol91].

Plusieurs sources de bruit peuvent affecter le circuit d'oscillateur : ses composants électroniques (transistors, résistances, …), les tensions d'alimentation, le substrat, etc. Ils apportent des fluctuations aléatoires aux tensions et aux courants du circuit. Ils peuvent être représentées par une source de tension ou de courant extérieure au circuit idéal comme illustré par la figure 4. 5. La pulsation de sortie de l'oscillateur ω(t) qui est fonction de la tension ou du courant d'entrée présente dans ce cas une erreur instantanée.

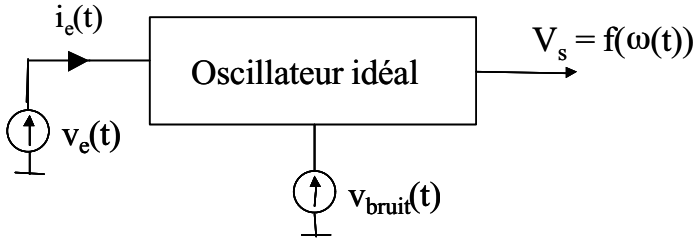

figure 4. 5 : présentation des sources de bruit dans un circuit d'oscillateur

En reprenant l'expression de la tension de sortie de l'oscillateur donnée par (4. 32), on peut définir la pulsation instantanée ω(t) comme suit :

$$\omega(t) = \frac{d}{dt}\left[\omega_0 t + \phi(t)\right] = \omega_0 + \dot{\phi}(t) \qquad (4.\ 35)$$

$\dot{\phi}(t)$ représente l'erreur de pulsation.

La source de bruit représentée par la figure 4.5 est caractérisée par sa densité spectrale de puissance. En supposant qu'elle présente des bruits blanc et en 1/f, l'allure de la densité spectrale de puissance de $\dot{\phi}$ est celle de la courbe de la figure 4. 6 :

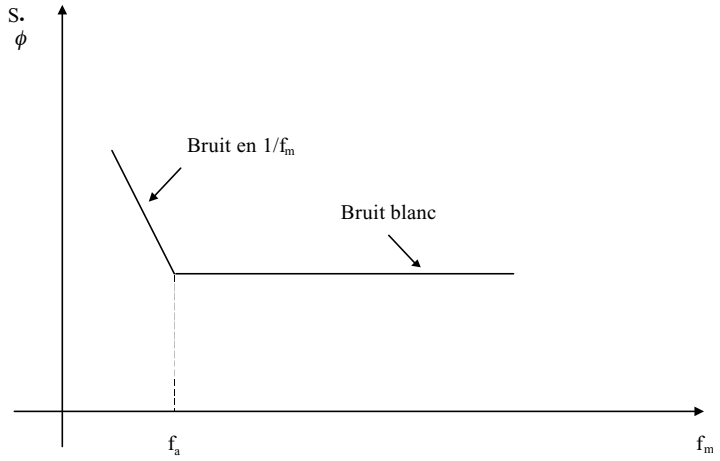

figure 4. 6 : allure de la densité spectrale de puissance des sources de bruit

$\dot{\phi}(t)$ est la dérivé temporelle de $\phi(t)$. Le passage d'une grandeur à l'autre dans le domaine fréquentiel correspond à une multiplication par jω et donc à une multiplication par ω^2 en terme de densité spectrale. Par conséquent $S_{\dot{\phi}}(\omega_m)$ et $S_\phi(\omega_m)$ sont reliées par la relation suivante [CuS66] [BaA66] :

$$S_{\dot{\phi}}(\omega_m) = \omega_m^2 \, S_\phi(\omega_m) \qquad (4.\ 36)$$

D'après l'allure de la densité spectrale de puissance de $\dot{\phi}(t)$ donnée par la figure 4. 6, la densité spectrale de puissance de la phase de sortie $S_\phi(f_m)$ est donc proportionnelle à $1/f_m^3$ pour les fréquences inférieures à f_a et à $1/f_m^2$ pour les fréquences supérieures à f_a comme l'illustre la figure 4. 7. A partir d'une fréquence f_b qui correspond à la limite de la bande passante de l'oscillateur, $S_\phi(f_m)$ devient constante [Lee66].

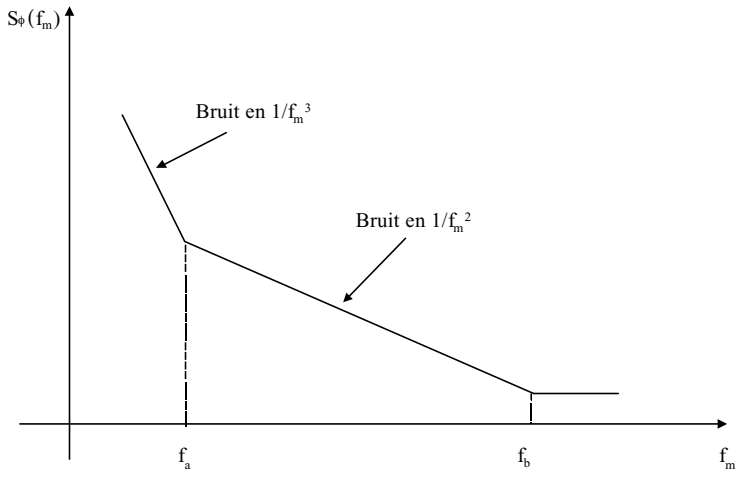

figure 4. 7 : allure de la densité spectrale de puissance de la phase ϕ(t)

III.2 Caractérisation du bruit de phase dans le domaine temporel

Dans le domaine temporel, le bruit de phase d'un oscillateur se traduit par une gigue (**jitter**) qui représente les fluctuations aléatoires de la période du signal de sortie. Pour un oscillateur en créneau par exemple, la gigue représente les fluctuations aléatoires des fronts montants et descendants comme illustré par la figure 4. 8.

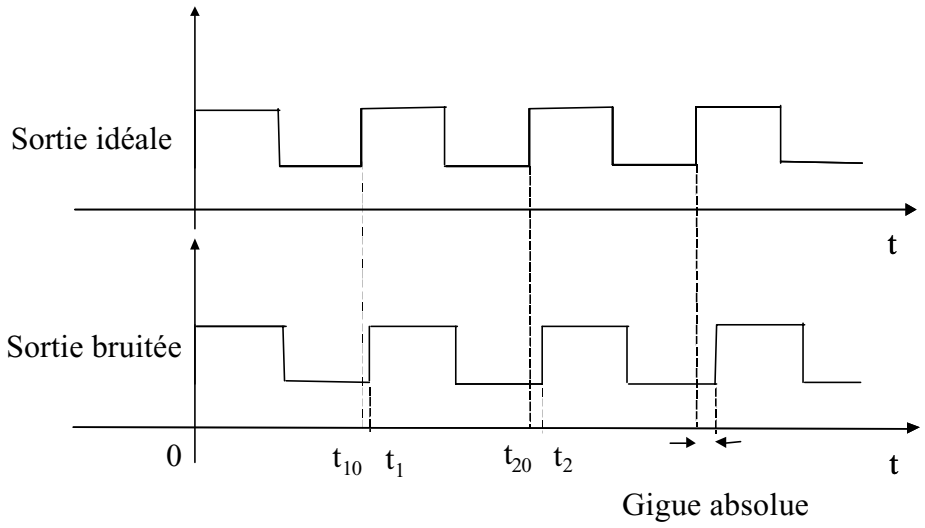

figure 4. 8 : illustration de la gigue absolue

Soit T_n la n^{ième} période du signal de sortie. On définit l'écart ΔT_n comme étant l'écart entre T_n et la période du signal non bruité $\overline{T}$:

$$\Delta T_n = T_n - \overline{T} \tag{4. 37}$$

Plusieurs définitions sont utilisées pour caractériser et mesurer la gigue. Elles sont définies par rapport à l'écart ΔT_n.

a) La gigue absolue

La gigue absolue ΔT_{abs} (**absolute jitter**) représente la somme des écarts ΔT_n [HeR99] :

$$\Delta T_{abs}(N) = \sum_{n=1}^{N} \Delta T_n \tag{4. 38}$$

La valeur de ΔT_{abs} à la période N représente l'écart entre la N^{ème} période de la sortie bruitée et celle de la sortie non bruitée comme le montre la figure 4. 8. Ce résultat peut être facilement démontré :

$$\begin{aligned} On\ a: \quad & t_2 - t_{20} = (t_2 - t_1) + (t_1 - 0) - t_{20} \\ & t_2 - t_{20} = T_2 + T_1 - 2\overline{T} \\ & t_2 - t_{20} = (T_2 - \overline{T}) + (T_1 - \overline{T}) \\ & t_2 - t_{20} = \Delta T_2 + \Delta T_1 \\ D'où: \quad & t_2 - t_{20} = \sum_{n=1}^{2} \Delta T_n = \Delta T_{abs}(2) \end{aligned}$$

$$Supposons\ que: \ t_N - t_{N0} = \sum_{n=1}^{N} \Delta T_n = \Delta T_{abs}(N),\ montrons\ que: \ t_{N+1} - t_{N+10} = \sum_{n=1}^{N+1} \Delta T_n = \Delta T_{abs}(N+1)$$

$$\begin{aligned} & t_{N+1} - t_{N+10} = t_{N+1} - t_N + (t_N - t_{N0}) + (t_{N0} - t_{N+10}) \\ & t_{N+1} - t_{N+10} = t_{N+1} - t_N - \overline{T} + \Delta T_{abs}(N) \\ & t_{N+1} - t_{N+10} = T_{N+1} - \overline{T} + \Delta T_{abs}(N) \\ & t_{N+1} - t_{N+10} = \Delta T_{N+1} + \Delta T_{abs}(N) \\ Donc \quad & t_{N+1} - t_{N+10} = \Delta T_{abs}(N+1) \end{aligned}$$

$$Par\ conséquent, on\ admet\ que: \quad t_N - t_{N0} = \sum_{n=1}^{N} \Delta T_n = \Delta T_{abs}(N)$$

$\Delta T_{abs}(N)$ représente la phase $\phi(t)$ du signal de sortie à la période N.

b) La gigue cyclique ΔT_c

La gigue cyclique ΔT_c (**cycle jitter**) est définie comme étant la valeur quadratique moyenne des écarts ΔT_n [HeR99] :

$$\Delta T_c = \lim_{N\to\infty} \sqrt{\frac{1}{N}\sum_{n=1}^{N} \Delta T_n^2} \qquad (4.\ 39)$$

c) La gigue cycle à cycle ΔT_{cc}

La gigue cycle-à-cycle ΔT_{cc} (**cycle to cycle jitter**) est la valeur quadratique moyenne de l'écart entre deux périodes successives [HeR98] :

$$\Delta T_{cc} = \lim_{N\to\infty} \sqrt{\frac{1}{N}\sum_{n=1}^{N} \left(T_{n+1} - T_n\right)^2} \qquad (4.\ 40)$$

ΔT_{cc} donne des informations sur la dynamique des fluctuations des périodes.

d) Relation entre ΔT_c et ΔT_{cc}

On définit la fonction d'auto corrélation de l'écart ΔT_n comme suit [HeR99] :

$$C_{\Delta T}(m) = \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \left(\Delta T_{n+m}\Delta T_n\right) \qquad (4.\ 41)$$

La gigue cyclique et la gigue cycle à cycle peuvent être définies en fonction de la fonction d'auto-corrélation par les expressions suivantes :

$$\Delta T_c^2 = \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \Delta T_n^2 = C_{\Delta T}(0) \qquad (4.\ 42)$$

$$et \quad \Delta T_{cc}^2 = \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \left(T_{n+1} - T_n\right)^2 = 2C_{\Delta T}(0) - 2C_{\Delta T}(1) \qquad (4.\ 43)$$

car $\quad T_{n+1} - T_n = \Delta T_{n+1} - \Delta T_n$

Lorsqu'il n'y a pas de corrélation entre les écarts ΔT_n, on obtient la relation simple suivante entre la gigue cyclique et la gigue cycle à cycle :

$$\Delta T_{cc} = \sqrt{2}\sqrt{C_{\Delta T}(0)} = \sqrt{2}\,\Delta T_c \qquad (4.\ 44)$$

Lorsque les sources de bruit sont supposées être des sources de bruit blanc, les écarts ΔT_n sont non corrélés. Dans ce cas, la relation (4. 44) est vérifiée [HeR99].

e) Définition des écarts de phase

Les grandeurs définies auparavant sont des temps qui s'expriment par conséquent en seconde. Pour les exprimer en radian, on définit l'écart $\Delta\phi_n$ qui s'exprime en fonction de l'écart ΔT_n par la relation suivante [Adl00] :

$$\Delta\phi_n = \Delta T_n \frac{2\pi}{\overline{T}} \qquad (4.\ 45)$$

On définit également la gigue de phase absolue $\Delta\phi_{abs}$, la gigue cyclique de phase $\Delta\phi_c$ et la gigue de phase cycle à cycle $\Delta\phi_{cc}$. Ils s'expriment respectivement en fonction de ΔT_{abs}, ΔT_c et ΔT_{cc} par les relations suivantes :

$$\Delta\phi_{abs} = \Delta T_{abs} \frac{2\pi}{\overline{T}} \qquad (4.\ 46)$$

$$\Delta\phi_c = \Delta T_c \frac{2\pi}{\overline{T}} \qquad (4.\ 47)$$

$$\Delta\phi_{cc} = \Delta T_{cc} \frac{2\pi}{\overline{T}} \qquad (4.\ 48)$$

III.3 Relations de passage entre grandeurs fréquentielles et temporelles

Nous allons considérer, pour la détermination des relations de passage entre les grandeurs fréquentielles et temporelles qui caractérisent le bruit de phase, que l'oscillateur n'est affecté que par des sources de bruit blanc. Nous supposons également l'ergodicité de nos variables aléatoires, ce qui est généralement le cas pour les signaux aléatoires physiques.

III.3.1 Calcul de l'écart type

ϕ(t) est une variable aléatoire. Pour cela, on s'intéresse généralement à la mesure de sa dispersion en faisant le calcul de son écart type $\sigma_{\phi(t)}$ ou de sa variance $\sigma^2_{\phi(t)}$ définie d'après la formule (4. 18) par l'expression suivante :

$$\sigma^2_{\phi(t)} = E\left[\phi^2(t)\right] - E^2\left[\phi(t)\right] \tag{4. 49}$$

L'erreur de pulsation $\dot{\phi}(t)$ est la dérivé par rapport au temps de la phase ϕ(t) :

$$\dot{\phi}(t) = \frac{d}{dt}\phi(t) \tag{4. 50}$$

D'où l'expression de ϕ(t) en fonction de $\dot{\phi}(t)$:

$$\phi(t) = \int_0^t \dot{\phi}(u)du + \phi(0) \tag{4. 51}$$

Puisque les sources de bruit sont supposées être des sources de bruit blanc, la densité spectrale de puissance $S_{\dot{\phi}}(\omega_m)$ de l'erreur de pulsation $\dot{\phi}(t)$ est constante, soit :

$$S_{\dot{\phi}}(\omega_m) = 2\ D_\phi \tag{4. 52}$$

D_ϕ représente la densité spectrale de puissance unilatérale de $\dot{\phi}(t)$.

Le produit d'auto-corrélation de $\dot{\phi}(t)$ est égal à la transformé inverse de Fourrier de $S_{\dot{\phi}}(w)$ (théorème de Wiener-Kinchine) [HeR99] :

$$C_{\dot{\phi}\dot{\phi}}(\tau) = \overline{\dot{\phi}(t+\tau)\,\dot{\phi}(t)} = 2\ D_\phi\ \delta(\tau) \tag{4. 53}$$

δ étant la fonction de Dirac.

Calculons la variance de la phase ϕ(t) ; d'après l'expression (4. 49) :

$$\sigma_\phi^2(t) = \overline{\left[\int_0^t \dot{\phi}(u)du\right]^2} - \overline{\int_0^t \dot{\phi}(u)du}^2$$

$$= \overline{\int_0^t \dot{\phi}(u)du \int_0^t \dot{\phi}(u')du'} - \left[\int_0^t \overline{\dot{\phi}(t)}du\right]^2$$

$$= \overline{\int_0^t\int_0^t \dot{\phi}(u)\dot{\phi}(u')dudu'} - \left[\int_0^t 0\,du\right]^2$$

$$= \int_0^t\int_0^t \overline{\dot{\phi}(u)\dot{\phi}(u')}\,dudu'$$

$$= \int_0^t \left(\int_0^t 2D_\phi\,\delta(u-u')\,du\right)du'$$

$$= 2D_\phi \int_0^t du' = 2D_\phi\, t$$

Nous obtenons donc l'expression suivante [TOK99] :

$$\sigma_\phi^2(t) = 2\,D_\phi\,t \tag{4. 54}$$

On en déduit l'expression de l'écart type σ_ϕ [DLF] [McN94]:

$$\sigma_\phi(t) = \sqrt{2\,D_\phi}\,\sqrt{t} \tag{4. 55}$$

En présence de sources de bruit blanc, l'écart type de la phase est proportionnel à la racine carrée du temps t. La valeur de $\phi(t)$ à la $N^{ème}$ période correspond à $\Delta\phi_{abs}(N)$. On déduit alors de (4. 55) l'expression de $\sigma_{\Delta\phi abs}(N)$:

$$\sigma_{\Delta\phi_{abs}}(N) = \sqrt{2\,D_\phi}\,\sqrt{N}\sqrt{\overline{T}} \tag{4. 56}$$

D'où d'après (4. 46) l'expression de $\sigma_{\Delta Tabs}(N)$:

$$\sigma_{\Delta T_{abs}}(N) = \frac{1}{\pi}\sqrt{\frac{D_\phi}{2}}\,\sqrt{N}\quad \overline{T}^{3/2} \tag{4. 57}$$

III.3.2 Calcul de la gigue cyclique

Dans le cas où N est égal à 1, calculons l'expression de la variance $\sigma^2_{\Delta\phi_{abs}}(1)$:

$$\sigma^2_{\Delta\phi_{abs}}(1) = \overline{\Delta\phi^2_{abs}(1)} = \overline{\Delta\phi_1^2}$$

L'écart $\Delta\phi$ est supposé ergodique, on a alors l'égalité entre la moyenne statistique et temporelle ; d'où :

$$\sigma^2_{\Delta\phi_{abs}}(1) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \Delta\phi_n^2 = \Delta\phi_c^2 = 2D_\phi \, \overline{T}$$

On obtient alors l'expression suivante de la gigue cyclique de phase [HeR99] :

$$\Delta\phi_c = \sqrt{2D_\phi} \sqrt{\overline{T}} \tag{4. 58}$$

On déduit l'expression de la gigue cyclique :

$$\Delta T_c = \frac{1}{\pi} \sqrt{\frac{D_\phi}{2}} \; \overline{T}^{3/2} \tag{4. 59}$$

Ce qui donne :

$$D_\phi = \frac{2\pi^2 \, \Delta T_c^2}{\overline{T}^3} \tag{4. 60}$$

Les sources de bruit sont supposées être des sources de bruit blanc, la gigue cycle à cycle s'exprime alors en fonction de la gigue cyclique selon la relation (4. 44). On aura finalement :

$$\Delta T_{cc} = \frac{1}{\pi} \sqrt{D_\phi} \; \overline{T}^{3/2} \tag{4. 61}$$

D'après les relations (4. 57), (4. 59) et (4. 61), on peut exprimer deux expressions qui lient l'écart type de la gigue absolue respectivement à la gigue cyclique et à la gigue cycle à cycle :

$$\sigma_{\Delta T_{abs}}(N) = \sqrt{N} \; \Delta T_c \tag{4. 62}$$

$$\sigma_{\Delta T_{abs}}(N) = \sqrt{\frac{N}{2}} \; \Delta T_{cc} \tag{4. 63}$$

III.3.3 Calcul de la densité spectrale de puissance

La densité spectrale de puissance de l'erreur de pulsation $S_{\dot{\phi}}(\omega_m)$ est constante :

$$S_{\dot{\phi}}(\omega_m) = 2\,D_\phi$$

En reprenant l'expression (4. 36) :

$$S_{\dot{\phi}}(\omega_m) = \omega_m^2\; S_\phi(\omega_m)$$

Il vient :

$$S_\phi(\omega_m) = \frac{2D_\phi}{\omega_m^2} \qquad (4.\ 64)$$

$S_\phi(\omega_m)$ est donc inversement proportionnelle au carré de ω_m.

III.3.4 Relation entre densité spectrale de puissance et gigue cyclique

En combinant les expressions (4. 59) et (4. 64), on peut exprimer la densité spectrale de puissance de la phase $S_\phi(\omega_m)$ en fonction de la gigue cyclique ΔT_c :

$$S_\phi(\omega_m) = \frac{1}{2\pi}\frac{\omega_0^3}{\omega_m^2}\,\Delta T_c^2 \qquad (4.\ 65)$$

En combinant (4. 61) et(4. 64), on déduit l'expression de $S_\phi(\omega_m)$ en fonction de la gigue cycle à cycle ΔT_{cc} :

$$S_\phi(\omega_m) = \frac{1}{4\pi}\frac{\omega_0^3}{\omega_m^2}\,\Delta T_{cc}^2 \qquad (4.\ 66)$$

IV Modélisation comportementale du bruit de phase

Notre objectif est de développer et de valider une méthode de modélisation comportementale du bruit de phase dans les oscillateurs dans le domaine temporel et ceci indépendamment de leur nature ou de leur architecture [FMD01]. Nos sources de bruit seront toutes des sources de bruit blanc. La méthode développée a été appliquée pour modéliser le bruit de phase dans les oscillateurs synchrones.

IV.1 Principe de la modélisation

Reprenons la figure 4. 8 qui montre l'effet du bruit sur la période de sortie :

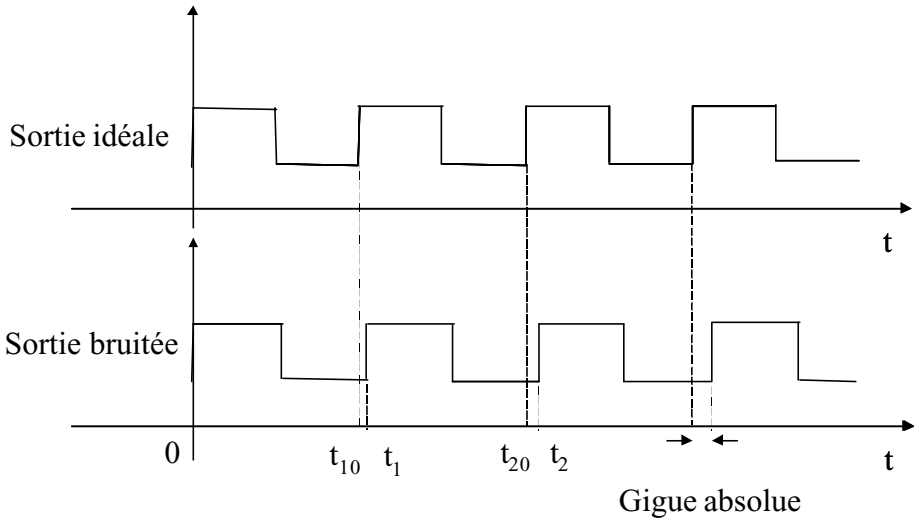

figure 4. 9 : effet du bruit de phase sur la sortie de l'oscillateur

La n^{ème} période T_n est égale à la période non bruitée $\overline{T}$ plus une petite variation ΔT_n. ΔT_n est un écart aléatoire qui dépend de la nature des sources de bruit affectant la sortie de l'oscillateur.

Nous allons considérer le cas d'une sortie en créneau. Le bruit de phase se traduit par les fluctuations des fronts montants et descendants. La demi-période s'exprime alors par l'expression suivante, déduite de (4. 37) :

$$\frac{T_n}{2} = \frac{\overline{T}}{2} + \frac{\Delta T_n}{2} \qquad (4.\ 67)$$

L'équation (4. 67) peut s'écrire aussi sous la forme suivante :

$$\frac{T_n}{2} = \frac{\overline{T}}{2}\left(1 + \delta T_n\right) \qquad (4.\ 68)$$

Avec $\delta T_n = \frac{\Delta T_n}{\overline{T}}$

δT_n est l'écart normalisé par rapport à la période. On peut l'exprimer sous la forme suivante :

$$\delta T_n = A_0 \, J_n \tag{4. 69}$$

A_0 est l'amplitude maximale de δT_n et J_n est un réel qui appartient à l'intervalle [-1, 1]. Il change aléatoirement de valeur à chaque période T_n.
Finalement, l'équation (4. 68) s'écrit :

$$\frac{T_n}{2} = \frac{\overline{T}}{2}\left(1 + A_0 \, J_n\right) \tag{4. 70}$$

Cette dernière équation sera utilisée pour définir la demi-période bruitée $\frac{T_n}{2}$ du signal de sortie des oscillateurs. Le réel J_n sera généré à chaque demi-période à partir de générateurs de bruit blanc.

IV.2 Générateurs de bruit blanc

Une source de bruit blanc génère une fonction aléatoire dont les valeurs sont non corrélées. Pour modéliser une telle source, nous allons considérer des fonctions qui génèrent des réels indépendants.

Nous avons alors utilisé le générateur pseudo aléatoire standard minimal proposé par Park et Miller [PaM] pour construire une fonction générant des entiers pseudo-aléatoires indépendants. Le générateur pseudo aléatoire standard minimal utilise la fonction de récurrence suivante :

$$x_n = 16807 \; x_{n-1} \bmod[2^{31}] \tag{4. 71}$$

Cette fonction génère des entiers compris entre -2^{31} et 2^{31}. En divisant ces entiers par 2^{31}, on génère des réels appartenant à l'intervalle [-1, 1].

Le langage VerilogA de CADENCE fournit dans sa bibliothèque mathématique la fonction **random** qui génère aléatoirement un entier à chaque appel et ADVance_MS de Mentor Graphics donne dans sa bibliothèque *math_real* la fonction **UNIFORM** qui génère aléatoirement des réels indépendants. Le tableau 4. 1 résume les principales caractéristiques de ces deux fonctions :

tableau 4. 1 : caractéristiques des fonctions random et UNIFORM

Fonction	**Standard minimal**	**random**	**UNIFORM**
Variable générée	Entier	entier	réel
Valeurs générées	$\left[-2^{31}, 2^{31}\right[$	$\left[-2^{31}, 2^{31}\right[$	$[0,1]$
Paramètres	_	seed	seed1, seed2
Type des paramètres	_	entier	variables positives

A chaque appel d'une de ces fonctions, un entier ou un réel aléatoire est généré ; il est indépendant du précédent. En changeant les valeurs des paramètres, on génère une combinaison aléatoire différente.

Nous allons utiliser ces différentes fonctions pour introduire des fluctuations aléatoires sur la demi-période de sortie.

IV.3 Introduction du générateur aléatoire dans les modèles d'oscillateurs

IV.3.1 Modèle VHDL-AMS

La figure 4. 10 présente un processus qui génère des réels aléatoires indépendants appartenant à l'intervalle [-1, 1] et ceci en faisant appel à la fonction **UNIFORM**. Le processus est activé à la fin de chaque demi-période bruitée, celle-ci étant calculée à partir de l'équation (4. 70). A chaque demi-période bruitée, un nouveau réel aléatoire est généré et par conséquent, une nouvelle demi-période bruitée est calculée. Les paramètres d'initialisation seed1 et seed2 permettent de choisir une suite de réels aléatoires ; leurs valeurs ont été choisis d'une façon arbitraire dans le processus ci-dessous :

```
t <= now ;
count <= t – t0 ;
P3 : process
        variable seed1 : positive := 19823;
        variable seed2 : positive := 124;
        variable x : real;
begin
        wait until count'above(half_ts_noised) = true;
        UNIFORM(seed1, seed2, x);
```

```
    Jn <= 2.0 * (x - 0.5);
    t0 <= t ;
end process P3;
half_ts_noised == 0.5 * ts * (1.0 + A*Jn);
```

figure 4. 10 : modèle en VHDL_AMS

Pour détecter la fin d'une demi-période bruitée, on utilise les variables **t**, $\mathbf{t_0}$ et **count** :

t : est le temps écoulé de l'analyse transitoire

$\mathbf{t_0}$: est le temps écoulé jusqu'à la dernière demi-période bruitée détectée

count : est un compteur qui calcule la différence entre **t** et $\mathbf{t_0}$ et qui est mis à zéro à la fin de chaque période bruitée.

IV.3.2 Modèles VerilogA

a) Utilisation de la fonction random

La figure 4. 11 présente un modèle VerilogA qui fait appel à la fonction **random** pour générer des réels aléatoires indépendants à chaque demi-période. Pour détecter la fin d'une demi-période bruitée, on utilise le même principe que celui utilisé dans le modèle VHDL-AMS. Le paramètre d'initialisation seed est choisi arbitrairement.

```
parameter integer seed = 72646 ;
    real t, count, t0 ;
    real Jn ;
    real half_ts_noised ;
Analog begin
t = $realtime ;
count <= t – t0 ;
if (count >= half_ts_noised)
begin
    Jn = random (seed)/33554432/64 ;
    half_ts_noised = 0.5*ts*(1 + A*Jn);
    t0 = t;
end
```

figure 4. 11 : modèle en VerilogA utilisant la fonction random

b) Utilisation de la fonction récurrente de Park et Miller

Dans le modèle de la figure 4. 12, nous définissons la fonction ***rnd*** qui applique la fonction de Park et Miller pour générer des réels aléatoires indépendants. Nous reprenons ensuite le même principe que précédemment pour faire varier aléatoirement la demi-période de sortie.

```
real n, n0 ;
real Jn ;
real half_ts_noised ;
real t, t0, count ;

fonction real rnd ;
        input x0 ;
        integer x0 ;
        real y ;
begin
        y = (16807*x0) % (2147483647) ;
        rnd = y ;
end
endfunction
analog begin
        n = rnd(n0) ;
        n0 = n ;
        Jn = n/33554432/64 ;
        t = $realtime ;
        count <= t – t0 ;
        if (count >= half_ts_noised)
        begin
        half_ts_noised = 0.5*ts*(1 + A*Jn);
        t0 = t;
        end
end
```

figure 4. 12 : modèle en VerilogA utilisant la fonction de Park et Miller

IV.4 Calcul des caractéristiques du bruit de phase

A fin de calculer les différentes grandeurs qui caractérisent le bruit de phase, nous avons développé des processus de calcul. Ces processus sont activés à la fin de chaque demi-période bruitée.

IV.4.1 Calcul de la gigue cyclique

La gigue cyclique est calculée à partir de sa définition donnée par l'expression (4. 39). Le processus *cycle_jitter* calcule à la fin de chaque demi-période bruitée la valeur du signal *cycle_jit* qui est défini à partir de l'expression suivante :

$$cycle\_jit = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\Delta T_n^2}$$

Voici le processus correspondant :

```
t <= now ;
count <= t – t0 ;
cycle_jitter : process
        variable DTn : real;
        variable DTn2 : real;
        variable sum_DTn2 : real := 0.0;
        variable N : real := 1.0;
begin
        wait until count'above(half_ts_noised) = true;
        DTn <= count – 0.5*ts;
        DTn2 <= DTn**2.0;
        sum_DTn2 <= sum_DTn2 + DTn2;
        cycle_jit <= sqrt(sum_DTn2/N);
        N <= N + 1.0;
        t0 <= t ;
end process cycle_jitter;
```

figure 4. 13 : calcul de la gigue cyclique

La figure 4. 14 donne l'allure du signal *cycle_jit* qu'on peut obtenir après une simulation temporelle :

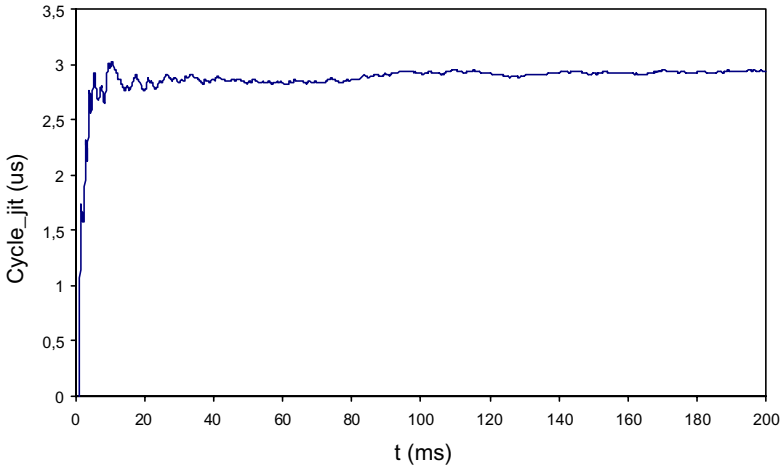

figure 4. 14 : allure du signal cycle_jit

La valeur de la gigue cyclique ΔT_c correspond à la limite de *cycle_jit* lorsque N tend vers l'infini. En pratique, cette valeur est atteinte après une centaine de périodes environ.

IV.4.2 Calcul de la gigue cycle à cycle

La gigue cycle à cycle est calculée à partir de sa définition donnée par l'expression (4. 40). Le processus *cycle_to_cycle_jitter* calcule à la fin de chaque demi-période bruitée la valeur du signal *cycle_to_cycle_jit* qui est défini à partir de l'expression suivante :

$$cycle\_to\_cycle\_jit = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(T_{n+1} - T_n\right)^2}$$

La figure 4. 15 présente le processus *cycle_to_cycle_jitter* :

```
t <= now ;
count <= t – t0 ;
cycle_to_cycle_jitter : process
        variable DTn1_n : real;
        variable DTn1_n2 : real;
        variable sum_DTn1_n2 : real := 0.0;
        variable count_n1 : real := 0.0;
        variable N : real := 1.0;
begin
        wait until count'above(half_ts_noised) = true;
        DTn1_n <= count – count_n1;
```

```
        DTn1_n2 <= DTn1_n**2.0;
        sum_DTn1_n2 <= sum_DTn1_n2 + DTn1_n2;
        cycle_to_cycle_jit <= sqrt(sum_DTn1_n2 / N);
        count_n1 <= count;
        N <= N + 1.0;
        t0 <= t ;
end process cycle_to_cycle_jitter;
```

figure 4. 15 : calcul de la gigue cycle à cycle

Le signal *cycle_to_cycle_jit* a pratiquement la même allure que celle du signal *cycle_jit*. La valeur de la gigue cycle à cycle ΔT_{cc} correspond à la limite de *cycle_to_cycle_jit* lorsque N tend vers l'infini. En pratique, cette valeur est atteinte après quelques centaines de périodes environ.

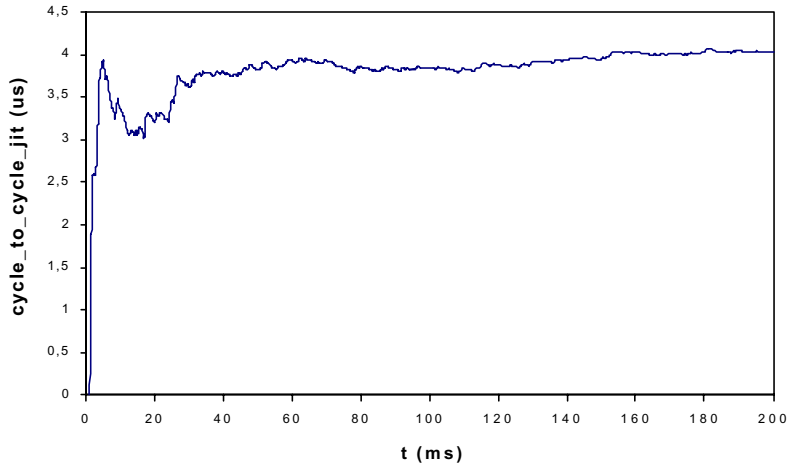

figure 4. 16 : allure du signal cycle_to_cycle_jit

IV.4.3 Calcul de la gigue absolue

La gigue absolue est calculée à partir de sa définition donnée par l'expression (4. 38). Le processus *absolute_jitter* calcule à la fin de chaque demi-période bruitée la valeur du signal *absolute_jit* qui est défini à partir de l'expression suivante :

$$absolute\_jit = \sum_{n=1}^{N} \Delta T_n$$

Voici le processus correspondant :

```
t <= now ;
count <= t – t0 ;
absolute_jitter : process
        variable DTn : real;
        variable absolute_jit : real := 0.0;
begin
        wait until count'above(half_ts_noised) = true;
        DTn <= count – 0.5*period;
        absolute_jit <= absolute_jit + DTn;
        t0 <= t ;
end process absolute_jitter;
```

figure 4. 17 : calcul de la gigue absolue

L'allure du signal *absolute_jit* est aléatoire, dépendant de la combinaison aléatoire donnée par le générateur de bruit. Pour deux combinaisons différentes, nous donnons dans la figure 4. 18 les deux allures correspondantes du signal absolute_jit :

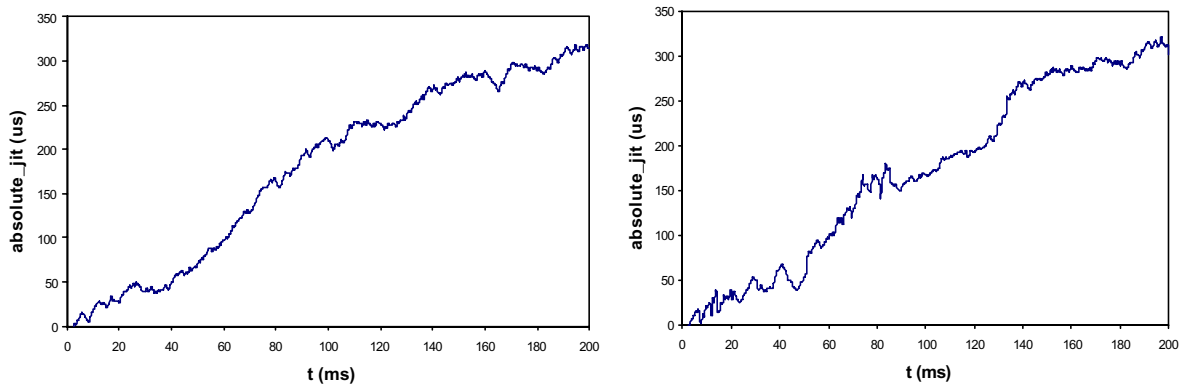

figure 4. 18 : allure du signal absolute_jit

L'information utile apportée par la simulation de la gigue absolue réside dans le calcul de l'écart type. Il faut alors obtenir toute une population de gigue absolue en faisant plusieurs simulations. Pour chaque simulation, on change de combinaison aléatoire du bruit injecté, ceci en agissant sur les paramètres d'initialisation des générateurs de bruits.

IV.4.4 Calcul de la densité spectrale de puissance

Il est possible de calculer la densité spectrale de puissance à partir de la relation qui la lie avec la gigue cyclique donnée par l'expression (4. 65) :

$$S_\phi(\omega_m) = \frac{1}{2\pi}\frac{\omega_0^3}{\omega_m^2}\Delta T_c^2$$

La figure 4. 19 présente un processus qui calcule $S_\phi(\omega_m)$. En faisant varier le paramètre ω_m, on peut construire le spectre de $S_\phi(\omega_m)$:

```
t <= now ;
count <= t – t0 ;
power_density : process
    variable Sphi : real;
begin
    wait until count'above(half_ts_noised) = true;
    If cycle_jit > 0.0 then
    Sphi <= 10.0*log10(cycle_jit* cycle_jit* cycle_jit / (ts*ts*ts*fm*fm));
    else Sphi <= 0.0;
    end if
    t0 <= t ;
end process power_density;
```

figure 4. 19 : calcul de la densité spectrale de puissance

IV.5 Validation de la méthode de modélisation

Pour valider notre méthode de modélisation, nous avons développé un modèle fonctionnel d'un oscillateur. La période de sortie de l'oscillateur est bruitée en appliquant l'expression (4. 70) suivante :

$$\frac{T_n}{2} = \frac{\overline{T}}{2}\left(1 + A_0 J_n\right)$$

J_n est obtenu en utilisant un générateur de bruit blanc.

Nous avons introduit dans le modèle les processus qui calculent les différentes grandeurs caractérisant le bruit de phase dans le domaine temporel qui sont la gigue absolue, la gigue cyclique et la gigue cycle à cycle. Nous allons déterminer par simulation comportementale ces grandeurs pour vérifier qu'elles respectent la théorie développée au début du chapitre.

IV.5.1 Simulation de la gigue absolue

La sortie de l'oscillateur est affectée par des sources de bruit blanc. L'écart type de la gigue absolue est donc proportionnel à la racine carrée du nombre de périodes N. Il s'exprime par l'expression suivante déduite de (4. 57) :

$$\sigma_{\Delta T_{abs}}(N) = \frac{1}{\pi}\sqrt{\frac{D_\phi}{2}}\sqrt{N}\left(\frac{\overline{T}}{2}\right)^{3/2} \qquad (4.\ 72)$$

Par rapport à l'expression (4. 57), la période $\overline{T}$ est remplacée par $\frac{\overline{T}}{2}$ du fait qu'on introduise les perturbations à chaque demi-période suivant l'expression (4. 70).

La valeur de la gigue absolue ΔT_{abs} dépend de la combinaison aléatoire des réels générés J_n. Pour obtenir toute une population de ΔT_{abs} permettant de calculer son écart type, on agit sur les paramètres d'initialisation du générateur aléatoire.

Le tableau suivant résume les valeurs des différents paramètres choisis pour les simulations.

tableau 4. 2 : paramètres de la simulation

Type de simulation	transitoire
Nombre de simulations	60
Période $\overline{T}$	1 ms
Amplitude de la gigue A_0	0.01

Pour différentes valeurs du nombre de période N, on calcule l'écart type de ΔT_{abs}. Le tableau 4. 3 résume les résultats trouvés :

tableau 4. 3 : calcul de l'écart type de la gigue absolue

N	40	80	120	160	200	240
$\sigma_{\Delta Tabs}$ (µs)	18.87	24.10	31.37	36.20	40.14	43.27

On trace alors la courbe $\sigma_{\Delta T_{abs}} = f(N)$:

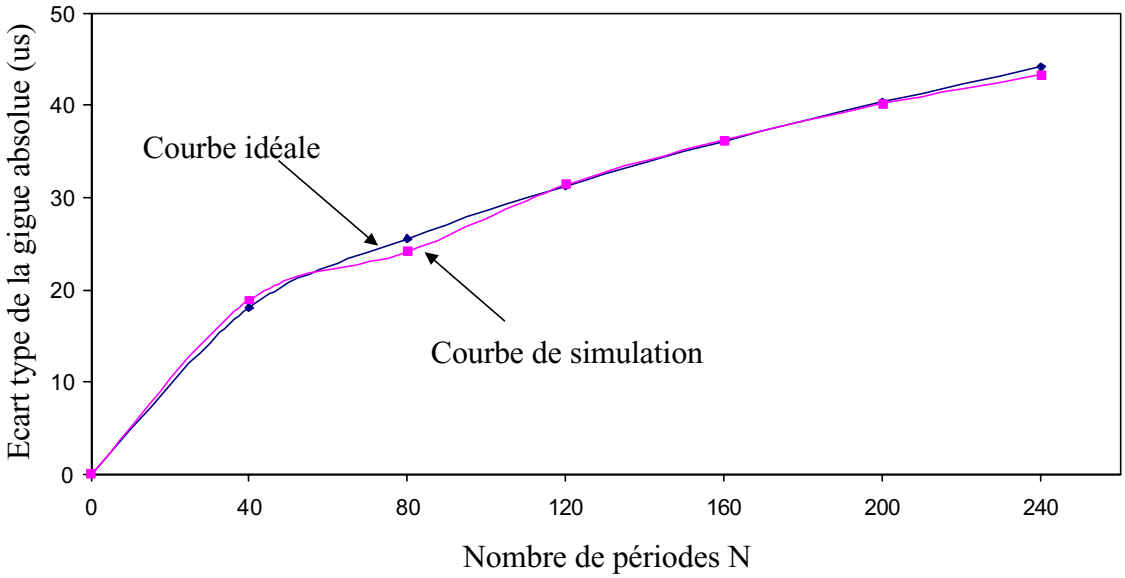

figure 4. 20 : courbe de l'écart type de la gigue absolue

Pour vérifier que $\sigma_{\Delta Tabs}$ est proportionnel à $\sqrt{N}$, la courbe obtenue est comparée à une courbe idéale de la forme $y = a\sqrt{N}$. Nous obtenons un bon accord entre cette courbe idéale et la courbe de simulation. On vérifie donc que le bruit injecté se comporte bien comme un bruit blanc ; ce qui valide notre méthode de modélisation. La détermination du coefficient a nous permet de calculer la densité D_ϕ par la formule : $a = \frac{1}{\pi}\sqrt{\frac{D_\phi}{2}}\left(\frac{\overline{T}}{2}\right)^{3/2}$ déduite de (4. 72).

A titre d'exemple, on trouve pour les paramètres de simulation choisis précédemment les valeurs suivantes :

tableau 4. 4 : détermination de D_ϕ

Paramètres	Valeurs numériques
a	2.84 e-6
D_ϕ	1.27

IV.5.2 Simulation de la gigue cyclique

La gigue cyclique est liée à l'écart type de la gigue absolue par la relation (4. 62) suivante :

$$\sigma_{\Delta T_{abs}}(N) = \sqrt{N}\ \Delta T_c$$

Nous avons vérifié par simulation que la valeur de ΔT_c est indépendante de la combinaison aléatoire du générateur de bruit blanc. Une seule simulation transitoire nous permet donc de déterminer ΔT_c.

En reprenant pour $\overline{T}$ et A_0 les mêmes valeurs qu'au paragraphe précédent, la simulation comportementale donne un ΔT_c égal à 2.836 µs. Cette valeur coïncide avec la valeur du coefficient *a* trouvée précédemment. La relation (4. 62) est donc vérifiée.

IV.5.3 Simulation de la gigue cycle àcycle

La gigue cycle à cycle est reliée à la gigue cyclique par l'expression (4. 44) suivante :

$$\Delta T_{cc} = \sqrt{2}\Delta T_c$$

En faisant varier la période $\overline{T}$, nous dressons le tableau suivant :

tableau 4. 5 : simulation de la gigue cycle à cycle

$\overline{T}$ (ms)	0.1	1	2	3	4	5	10
ΔT_{cc} (µs)	0.387	3.873	7.747	12.52	15.49	20.33	40.66
ΔT_c (µs)	0.284	2.836	5.673	9	11.35	14.82	29.64
$\Delta T_{cc}/\Delta T_c$	**1.36**	**1.37**	**1.37**	**1.39**	**1.36**	**1.37**	**1.37**

Pour les différentes valeurs de la période $\overline{T}$, on vérifie en pratique un rapport entre ΔT_{cc} et ΔT_c égal à $\sqrt{2}$ à 3% près.

IV.6 Paramètre d'ajustement du bruit de phase

Le paramètre A_0 permet d'ajuster l'amplitude maximale de la fluctuation de la période de sortie. Nous allons étudier son influence sur la gigue cyclique et la gigue cycle à cycle.

En faisant varier la valeur du paramètre A_0, on relève ΔT_{cc} et ΔT_c. Le calcul du rapport $\frac{\Delta T_{cc}}{\Delta T_c}$ nous a permis de vérifier que A_0 n'affecte pas le rapport $\sqrt{2}$ qui existe entre ces deux grandeurs. Les résultats obtenus sont regroupés dans le tableau 4. 6, $\overline{T}$ étant égale à 1ms.

tableau 4. 6 : influence du paramètre A_0

A_0	0.01	0.008	0.006	0.004	0.002
ΔT_{cc} (µs)	3.873	3.163	2.396	1.66	0.83
ΔT_c (µs)	2.836	2.297	1.728	1.177	0.60
$\frac{\Delta T_{cc}}{\Delta T_c}$	1.37	1.38	1.39	1.41	1.38

Nous traçons la courbe $\Delta T_c = f(A_0)$:

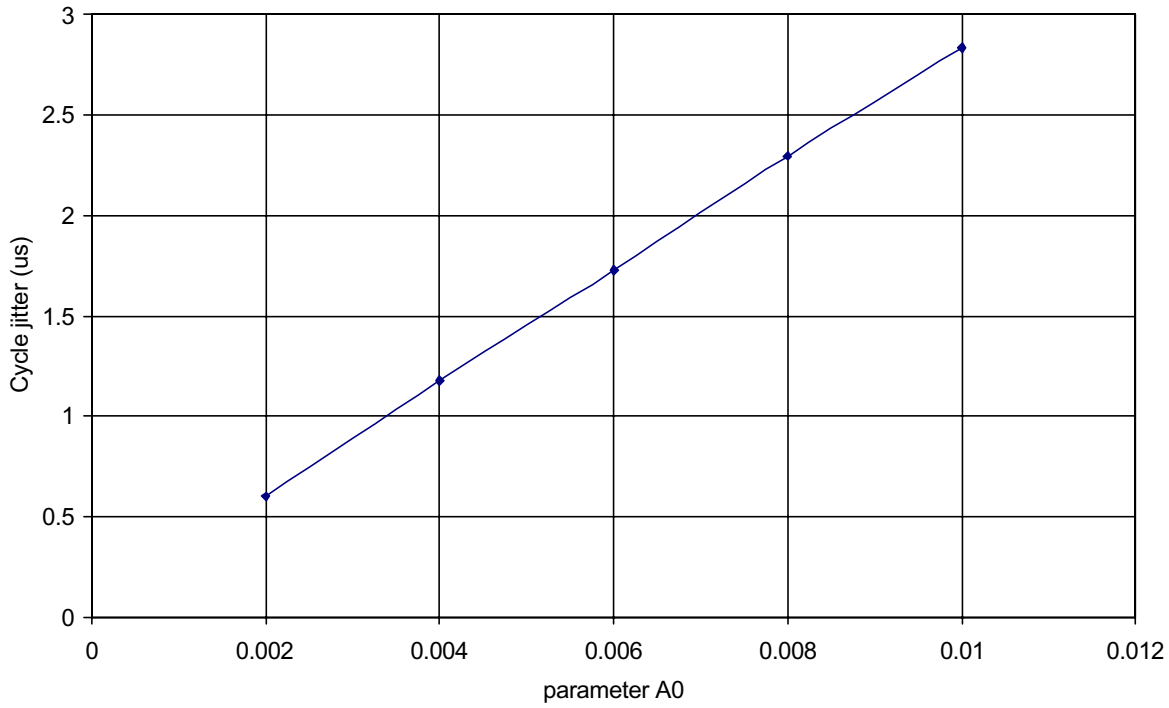

figure 4. 21 : évolution de la gigue cyclique avec le paramètre A_0

La courbe obtenue est linéaire donc la gigue cyclique et par conséquent la gigue cycle à cycle sont directement proportionnelles au paramètre A_0.

V Modélisation du bruit de phase dans les oscillateurs synchrones

En l'absence de signal de synchronisation, l'oscillateur synchrone présente une caractéristique de bruit de phase qui lui est propre. Quand il est synchronisé, il recopie le bruit de phase du signal d'entrée au facteur de synchronisation près [Bad00].

V.1 Modélisation du bruit de phase

Pour modéliser le bruit de phase de l'oscillateur synchrone, nous ajoutons au modèle décrit dans le chapitre précédent de nouveaux processus qui sont :

- Le processus de génération d'un bruit blanc décrit par la figure 4. 10.
- Les processus de calcul de la gigue cyclique, de la gigue cycle à cycle et de la gigue absolue décrits respectivement par les figures 4.13, 4.15 et 4.17.
- Le processus de calcul de la densité spectrale de puissance décrit par la figure 4. 19.

Pour introduire une fluctuation à la période de sortie T_{sortie}, nous appliquons l'expression suivante, déduite de l'équation (4. 70) :

$$\frac{T_{sortie}}{2} = \frac{1}{2\,f_s}\quad\left(1 + A_0\;\; Jitter\right) \tag{4. 73}$$

f_s étant la fréquence de sortie et A_0 l'amplitude maximale de la fluctuation.

Puisque le bruit de phase de l'oscillateur synchrone est différent selon qu'il est ou n'est pas synchronisé, nous introduisons une nouvelle variable *jitter*. Sa valeur dépend de l'état de l'oscillateur.

a) Oscillateur synchronisé

Si l'oscillateur est synchronisé, la valeur de la fréquence de sortie f_s est égale à la fréquence d'entrée f_{in} multipliée par l'harmonique de synchronisation n. La valeur de la variable *jitter* est prise égale à zéro. L'expression de la période de sortie devient :

$$\frac{T_{sortie}}{2} = \frac{1}{2\,f_s} = \frac{1}{2\,n\,f_{in}} = \frac{T_{in}}{2\,n} \tag{4. 74}$$

L'oscillateur recopie alors le bruit du signal d'entrée au facteur de synchronisation près.

b) Oscillateur non synchronisé

Si l'oscillateur n'est pas synchronisé, la fréquence de sortie f_s est égale à sa fréquence propre f_0. La valeur de la variable *jitter* prend la valeur de sortie du générateur de bruit blanc J_n. L'expression de la période de sortie devient alors :

$$\frac{T_{sortie}}{2} = \frac{1}{2 f_0} \left(1 + A_0 \ J_n\right) = \frac{T_0}{2} \left(1 + A_0 \ J_n\right) \qquad (4.\ 75)$$

En ajustant le paramètre A_0, on peut retrouver la caractéristique du bruit de phase de l'oscillateur non synchronisé.

La valeur de la variable *jitter* est déterminée dans le processus *synchro* dans lequel est fait le test de synchronisation :

```
finf * n == f0 - 0.5*df;
fsup * n == f0 + 0.5*df;
synchro : process
begin
        wait until count'above(half_ts_noised) = true;
        if fin > finf and fin < fsup then
        fs <= fin * n;
        jitter <= 0.0;
        else fs <= f0;
        jitter <= Jn;
        end if;
end process synchro;
```

figure 4. 22 : processus de test de synchronisation

V.2 Simulation du bruit de phase

En l'absence de signal de synchronisation, le bruit de phase de l'oscillateur synchrone COLPITTS 2.4 GHz dont les caractéristiques ont été décrites dans le chapitre précédent a été simulé [FMB01]. En faisant varier la fréquence d'offset f_m, on obtient la caractéristique du bruit de phase de l'oscillateur libre.

La même caractéristique a été simulée au niveau transistor en utilisant le simulateur spectreRF de CADENCE. Des mesures de bruit ont été également réalisées sur le prototype fabriqué.

Pour comparer les résultats obtenus, nous représentons sur le même graphe les trois caractéristiques obtenues. Remarquons qu'il s'agit de la caractéristique de la densité spectrale de puissance unilatérale $L(f_m)$

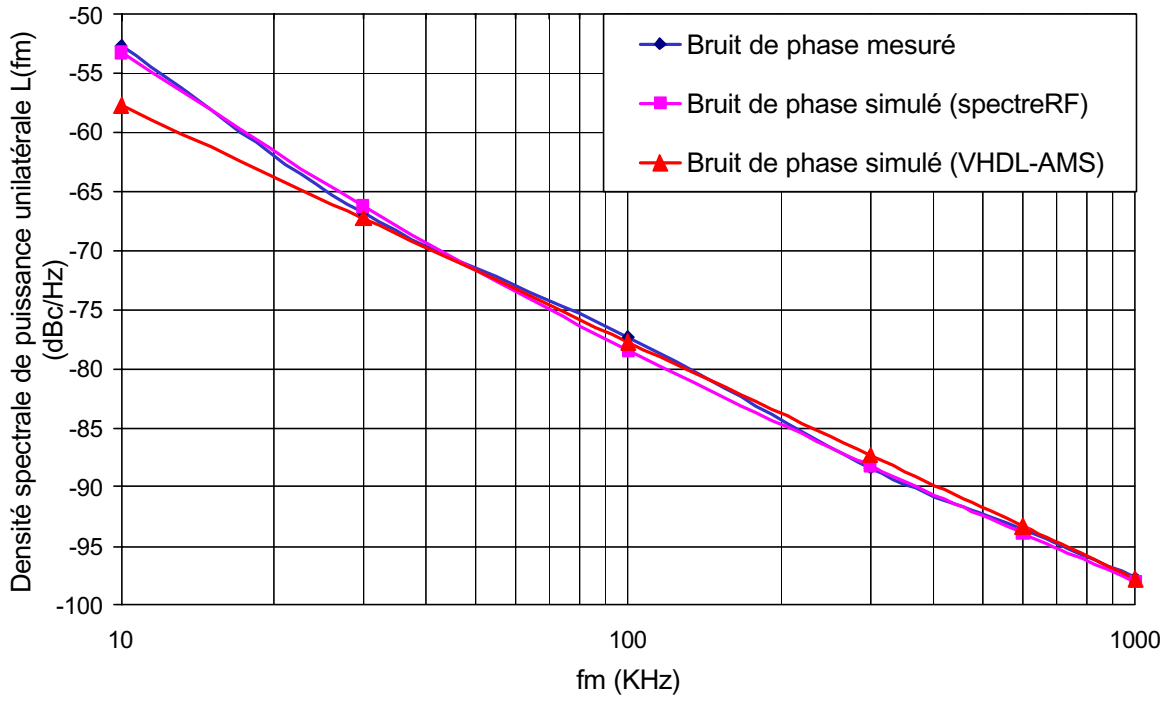

figure 4. 23 : Caractéristique du bruit de phase de l'oscillateur libre

On remarque tout d'abord le bon accord des mesures avec la simulation spectreRF. A partir d'une fréquence d'offset de l'ordre de 30 KHz, il y a aussi un bon accord des mesures avec la simulation comportementale.

Les mesures montrent que le bruit de phase de l'oscillateur est proportionnel à $1/f^3$ pour les faibles fréquences d'offset. A partir d'une fréquence de l'ordre de 30 KHz, il devient proportionnel à $1/f^2$. Ceci vérifie l'allure du bruit dans les oscillateurs donnée au début de ce chapitre.

Dans le modèle de l'oscillateur synchrone développé, nous ne tenons compte que des sources de bruit blanc. Ceci explique l'allure en $1/f^2$ de la caractéristique du bruit de la simulation comportementale. Pour que cette caractéristique soit proportionnelle à $1/f^3$ pour les faibles fréquences d'offset, il faut introduire avec la source de bruit blanc, une source de bruit en 1/f.

La méthode de modélisation du bruit de phase dans l'oscillateur synchrone peut s'appliquer aussi à d'autres types d'oscillateurs. Le paramètre A_0 permettra d'ajuster la caractéristique de bruit par rapport à des mesures ou simulations faites au niveau transistor.

VI Conclusion

Dans ce chapitre, nous avons étudié le bruit de phase dans les oscillateurs en commençant par donner des généralités sur les signaux aléatoires. Nous avons étudié les caractéristiques du bruit de phase dans les deux domaines fréquentiel et temporel. Nous avons ensuite développé les relations de passage entre grandeurs fréquentielles et temporelles en supposant que l'oscillateur n'est affecté que par des sources de bruit blanc.

Nous avons également détaillé notre démarche pour modéliser le bruit de phase dans le domaine temporel. En appliquant cette démarche, nous avons modélisé le bruit de phase dans les oscillateurs synchrones et nous avons comparé les résultats de simulations comportementales avec les simulations niveau transistor et les mesures.

De nouvelles perspectives se présentent : premièrement la modélisation du bruit en 1/f et deuxièmement la modélisation du bruit de phase des autres blocs constituant les circuits synthétiseurs de fréquence, principalement le comparateur de phase et les diviseurs de fréquences. L'objectif sera de modéliser le bruit de phase dans ce type de circuits en tenant compte des bruits blancs et des bruits en 1/f.

Conclusion générale et perspectives

En ce début du 21ème siècle, les nouvelles tendances de l'électronique sont l'intégration des System-on-Chip (SoC) et des ASIC mixtes comprenant, sur une surface toujours plus réduite, des fonctions de plus en plus complexes.

Les outils de CAO analogiques doivent évoluer pour rattraper leur retard sur la conception numérique qui est de nos jours largement automatisée. Cette automatisation est devenue possible grâce au développement des langages de description matérielle standards tel que le VHDL associé aux outils de synthèse numérique, et surtout à la nature même des circuits numériques pour lesquels les niveaux inférieurs sont relativement figés.

L'extension de ces standards pour les circuits analogiques et mixtes (norme VHDL-AMS par exemple) ouvre la voie à l'amélioration des outils de conception analogique et mixte et peut être au développement des outils de synthèses analogiques et mixtes. Le seul succès de ces langages standards ne répond pas aux besoins de la conception analogique et mixte. L'objectif est maintenant de donner aux concepteurs les moyens méthodologiques et techniques leur permettant d'appliquer la méthode de conception hiérarchique Top-Down pour les circuits analogiques et mixtes.

L'aspect méthodologique est abordé au chapitre 2 à laquelle on a tenté de répondre à la question : 'Comment procède-t-on pour concevoir un modèle comportemental?'. Nous avons alors présenté deux approches de modélisation : une approche schématique basée sur l'exploitation du schéma transistor et une approche fonctionnelle qui analyse la fonction du circuit. L'approche schématique est plutôt adaptée à la phase Bottom-Up pour extraire des

modèles raffinés, alors que l'approche fonctionnelle est utilisée dans la phase Top-Down durant laquelle on commence par définir les spécifications du système à concevoir.

On s'aperçoit que dans un modèle fonctionnel, des sous fonctions apparaissent fréquemment. Nous les avons recensées et nous avons décomposé par conséquent nos modèles en trois parties : une partie détection des variables d'entrée ; une deuxième partie pour le calcul des paramètres des signaux de sortie et une dernière partie pour générer les signaux de sortie. Cette démarche systématique nous a permis d'uniformiser l'architecture de nos modèles et de faire appel à des sous blocs que l'on retrouve dans plusieurs modèles.

Un des freins principaux à l'évolution des outils de synthèse analogique et mixte est l'absence de bibliothèques de modèles standards. En se limitant au domaine radio-fréquence et plus précisément à la synthèse de fréquence, nous avons développé une bibliothèque de modèle RF que nous avons pris le soin de bien documenter pour qu'elle soit accessible et compréhensible par les autres utilisateurs.

Pour valider notre bibliothèque de modèles RF, nous avons développé des modèles pour trois applications : un oscillateur synchrone 2.4 GHz de type COLPIITS dédié aux réseaux locaux sans fil, un oscillateur synchrone 5.2 GHz à résistance négative dédié aux applications de type HIPERLAN et un synthétiseur de fréquence fractionnaire répondant à la norme UMTS'2000. Les performances des modèles vérifient les résultats de mesures et des simulations au niveau transistor.

Grâce aux temps de simulation très courts des modèles comportementaux, cette bibliothèque permet d'explorer rapidement les architectures de synthétiseurs et de faire une première optimisation des paramètres de haut niveau.

Pour les applications radio-fréquence, il est important d'étudier et d'optimiser le bruit de phase. Nous avons alors étudié dans le chapitre 4 le bruit de phase dans les oscillateurs en commençant par donner des généralités sur les signaux aléatoires. Nous avons étudié les spécifications du bruit de phase dans les domaines fréquentiel et temporel en considérant seulement le cas de sources de bruit blanc.

Nous avons alors développé une démarche systématique pour introduire l'effet du bruit de phase dans nos modèles d'oscillateurs, en particulier l'oscillateur synchrone. Là

encore, les modèles ont été validés par comparaison avec des résultats de mesures, réalisés sur prototypes.

A la suite des travaux de cette thèse, divers développements sont envisagés :

- Tous les modèles présentés dans cette thèse, et dont certains sont fournis en annexe, doivent faire l'objet d'une procédure de qualification visant à les rendre utilisables par tous. Ils seront alors déposés en libre accès sur un site Internet[1].

- Il conviendrait d'étendre cette bibliothèque à d'autres blocs participant aux chaînes d'émission et de réception des circuits de télécommunication. Une étude est en cours concernant le développement de modèles de modulateurs Σ Δ.

- Dans nos modèles d'oscillateurs bruités, seules les sources de bruit blanc ont été considérées. Or, il apparaît qu'aux faibles fréquences, les sources de bruit en 1/f doivent être prises en compte. La démarche de modélisation du bruit de phase doit donc être adaptée et rendue plus générale.

- La modélisation du bruit de phase devra être également considérée pour d'autres blocs constituant les systèmes de synthèse de fréquence tel que les comparateurs de phase ou les diviseurs de fréquence.

[1] http://www.beams.asso.fr

Références bibliographiques

[AbM83] A. A. Abidi, R. G. Meyer, *Noise in relaxation oscillators, IEEE journal of solid-state circuits*, Vol. sc-18, N° 6, décembre 1983.

[Adl00] J. Adler, *Jitter in clock sources*, Vectron International, 2000.

[ADV00] : *ADVance_MS reference manual*, Mentor Graphics, 2000.

[Ald01] : P. Aldebert, *Le langage VHDL-AMS - modélisation et simulation des systèmes analogiques et mixtes*, http://www.supelec.fr/fc/eg3.html.

[AnB94] B. A. A. Antao, A. J. Brodersen, *Behavioral simulation for analog system design verification*, IEEE 1994 custom integrated circuits conference, pp 21.2.1-21.2.4.

[AnB96] B. A. A. Antao, A. J. Brodersen, *A formulation for analog behavioral level sensitivity and steady state analysis*, Analog integrated circuits and signal processing, Novembre 1996, pp 217-229.

[AnS95] B. A. A. Antao, R. Saleh, *Simulation model transformations for analog hardware description languages*, IEEE 1995 custom integrated circuits conference, pp 25.6.1-25.6.4.

[Ant95] B. A. A. Antao, *Architectural exploration for analog system synthesis*, IEEE 1995 custom integrated circuits conference, pp 25.4.1-25.4.4.

[Ant96] B. A. A. Antao, *AHD Languages – A must for time-cristal designs*; Circuits and devices, Juillet 1996.

[ARH96] M. M. Ahmed, H. F. Ragaie, H. Haddara, *A hierarchical approach to the analog behavioural modeling of neural networks using HDL-A*, IEEE 1996.

[BaA66] J. A. Barnes, D. W. Allan, *A statistical model of Flicker noise*, Proceedings of the IEEE, Vol 54, N°2, pp 176-178, février 1966.

[Bad00] F. Badets, *Contribution à l'étude de la synchronisation des oscillateurs: intégration des oscillateurs synchrones dans les systèmes radiofréquences en technologie Silicium*, thèse, Université Bordeaux 1, Janvier 2000.

[BHB96] C. Borchers, L. Hedrich, E. Barke, *Equation-based behavioural model generation for non-linear analogue circuits*, ACM IEEE Design Automation Conference, Las Vegas, 1996.

[BoP92] P. Bolcato, R. Poujois, *A new approach for noise simulation in transient analysis*, IEEE international symposium circuits and systems ISCAS'92, Vol. 25, pp 887-890, mai 1992.

[Cad97] : *Spectre Reference manual*, Cadence, 1997.

[CaS91] G. Casinovi, A. Sangiovanni-Vincentelli, *A macromodeling algorithm for analog circuits*, IEEE transactions on computer aided design, Vol. 10, N° 2, pp 150-160, février 1991.

[CBD99] : E. Christen, K. Bakalar, A. M. Dewey, E. Moser, *Analog and Mixed Signal Modeling Using the VHDL-AMS Language*, 36th Design Automation Conference New Orleans, Juin 21-25, 1999.

[Cha90] M. Charbit, *Eléments de théorie du signal : les signaux aléatoires*, collection Pédagogique de Télécommunication, 1990.

[Com01] : *Company Info*, http://www.cadence.com/company.

[Cou01] : B. Courtois, *Quelques tendances en microélectronique*, issu de

http://www.ensem.u-nancy.fr/Microlor/bulletin/bul12.pdf.

[CuS66] L. S. Cutler, C. L. Searle, *Some aspects of the theory and measurement of frequency fluctuations in frequency standards*, Proceedings of the IEEE, Vol 54, N°2, pp 136-154, février 1966.

[Dab95] J. Dabrowski, *Functional-Level analog macromodeling with piecewise linear signals*, Proceeding Design Automation Conference with EURO VHDL; 1995 San Fransisco ca Moscone center; pp 222-227, 1995.

[DaP98] J. Dabrowski, A. Pulka, *Approach to PWL analogue modelling in VHDL environment*, Analogue integrated circuits and signal processing, Vol N° 16, Boston, 1998.

[DaS94] Z. Daoud, C. J. Spanos, *DORIC: design of optimal and robust integrated circuits*, IEEE 1994 custom integrated circuits conference, pp 15.3.1-15.3.4.

[Dev94] Y. Deval, *Conception de circuits intégrés analogiques – Elaboration d'une méthode originale d'optimisation basée sur la technique des plans d'expériences*, thèse doctorat, Université Bordeaux 1, 1994.

[DFD96] L. Dubois, T. Fukuda, F. Delmotte, P. Borne, *Multi-model systems are universal approximators*, Symposium on modelling analysis and simulation CESA'96 IMACS multiconference, pp 879-884, France, Juillet, 1996.

[DLF] P. Delatte, J. D. Legat, D. Flandre, *PLL Jitter Behavioural modelling and simulation using HDL-A*,

[DND99] A. Doboli, A. Nuez-Aldana, N. Dhanwada, S. Ganesan, R. Vemuri, *Behavioral synthesis of analog systems using two layered design space exploration*, DAC 1999, New Orleans, Louisiana, pp 951-957, 1999.

[DSG94] S. Donnay, K. Swings, G. Gielen, W. Sansen, *A methodology for analog high-level synthesis*, IEEE 1994 custom integrated circuits conference, pp 15.6.1-15.6.4.

[FMB02] A. Fakhfakh, N. Milet-Lewis, J.B. Bégueret, H. Lévi, *VHDL-AMS model of a synchronous oscillator including phase noise*, System on Chip Design Languages (Best of FDL'01 and HDLCon'01), Anne Mignotte, Eugenio Villar et Leslie Spruiell (Eds.), CHDL Series, Kluwer, 2002.

[FMB01] A. Fakhfakh, N. Milet-Lewis, J.B. Bégueret, H. Lévi, *VHDL-AMS model of a synchronous oscillator including phase noise*, FDL'01, Lyon, Septembre 2001.

[FMD01] A. Fakhfakh, N. Milet-Lewis, Y. Deval, H. Lévi, *Study and behavioural simulation of phase noise and jitter in oscillators*, ISCAS'2001, Sydney, Mai 2001.

[FML00] A. Fakhfakh, N. Milet-Lewis, H. LEVI, *Etude et simulation comportementale du bruit de phase et du jitter des oscillateurs*, Proceeding des Journées Scientifiques Franco-Tunisiennes 2000, pp 137-141, Monastir, Tunisie, octobre 2000.

[FMZ00] A. Fakhfakh, N. Milet-Lewis, T. Zimmer, H. LEVI, *a behavioural modelling procedure for analogue integrated circuits – application to PLL behavioural simulation*, proceeding of the 2000 international conference on artificial intelligence for decision – control and automation in engineering and industrial applications, Vol. Engineering and industrial applications, Monastir, Tunisie, mars 2000.

[Gar83] C. W. Gardiner, *Handbook of stochastic methods*, Berlin: Springer-Verlag, 1983.

[GHH95] M. Goedecke, H. Hamad, S. A. Huss, *A methodology for the development of system-level simulation models for analog functional blocks*, Archiv für Electronik und Ubestragungrteshnik, Vol. 49, N° 2, pp 72-80, 1995.

[Gre94] A. B. Grebene, *A methodical approach for macro modelling of analogue circuits*, Nortcon'94 conference record, pp 250-210, 1994.

[GRS98] N. J. Godambe, C. J. Richard Shi, *Behavioural level noise modeling and jitter simulation of phase-locked loops with faults using VHDL-AMS*, Journal of electronic testing: theory and applications 13, pp 7-17, 1998.

[GuE95] R. S. Guindi, M. Elmasry, *High level analog synthesis using signal flow graph transformations*, 1995 IEEE, pp 366-369.

[GuH96] S. K. Gupta, M. M. Hassan, *KANSYS : a CAD tool for analog circuit synthesis*, 9th international conference on VLSI design, pp 333-334, Janvier 1996.

[Haf66] E. Hafner, *The effects of noise in oscillators*, Proceedings of the IEEE, Vol 54, N°2, pp 179-198, février 1966.

[Haj99] A. Hajimiri, *The design of low noise oscillators*, 1999.

[HaL98] A. Hajimiri, T. H. Lee, *A general theory of phase noise in electrical oscillators*. IEEE journal of solid-state circuits, Vol. 33, N° 2, février 1998.

[Her01] : Y. Hervé, *VHDL-AMS un langage pour l'électronique du futur*, Stage CNFM du 17 au 20 avril 2001, Migrest-Strasbourg.

[HeR98] F. Herzel, B. Razavi, *Oscillator jitter due to supply and substrate noise*, IEEE custom integrated circuits conference, 1998.

[HeR99] F. Herzel, B. Razavi, *A study of oscillator jitter due to supply and substrate noise*, IEEE transactions on circuits and systems- II: analog and digital signal processing, Vol. 46, N° 1, janvier 1999.

[HHN94] R. K. Henderson, M. Hinners, P. Nussbaum, L. Asier, *Capture and re-use of analog simulation knowledge*, IEEE 1994 custom integrated circuits conference, pp 15.2.1-15.2.4.

[HLL99] A. Hajimiri, S. Limotyrakis, T. H. Lee, *Jitter and phase noise in ring oscillators*. IEEE journal of solid-state circuits, Vol. 34, N° 6, Juin 1999.

[Hol00] K. Holladay, *Design a PLL for specific loop bandwidth*, END EUROPE, pp 64-66, october 2000, http//:www.ednmag.com.

[HuW47] R. Huntoon, A. Weiss, *synchronisation of oscillators*, proceedings of the I.R.E., Vol. 35, pp 1415-1423, décembre 1947.

[IGC00] S. Ishioka, Z. Gingl, D. Choi, N. Fuchikami, *Amplitude truncation of Gaussian $1/f^{\alpha}$ noises*, Physics Letters A 269, pp 7-12, avril 2000.

[Jel96] I. Jelinski, *Traitement du signal – Asservissements linéaires*, série Vuibert Technologie, 1996.

[Kas95] N. J. Kasdin, *Discrete simulation of colored noise and stochastic processes and $1/f^{\alpha}$ power law noise generation*, Proceedings of the IEEE, Vol. 83, N° 5, pp 802-827, Mai 1995.

[KlM95] D. J. Klein, M. L. Manwaring, *A differential model approach to analog design automation*, 1995 IEEE, pp 22-27.

[Kun98] K. Kundert, *Modelling and simulation of jitter in PLL – Frequency synthesisers*, Cadence confidential manuscript, décembre 98.

[KWG] B. Kim, T. C. Weigandt, P. R. Gray, *PLL/DLL system noise analysis for low jitter clock synthesizer design*, pp 31-34.

[LCK97] K. Lim, S. Choit, B. Kim, *Optimal loop bandwidth design for low noise PLL applications*, 1997 IEEE, pp 425-428.

[Lee66] D. B. Leeson, *A simple model of feedback oscillator noise spectrum*, Proceedings Letters, pp 329-330, février 1966.

[MaA93] H. A. Mantooth, P.E. Allen, *A higher level modelling procedure for integrated circuits*, Analogue Integrated Circuits and Signal Processing 3, Vol N°3, Boston, 1993.

[MaE97] F. Manneville, J. Esquieu, *Electronique-Théorie du signal et composants*, Dunod, Paris, 1997.

[Mar01] B. Martin, *Automation comes to analog*, IEEE Spectrum, pp 70-75, Juin 2001.

[McN94] J. A. McNeill, *Jitter in ring oscillators. IEEE international symposium circuits and systems ISCAS'94*, pp 201-204, June 1994.

[McN97] J. A. McNeill, *Jitter in ring oscillators IEEE journal of solid-state circuits*, vol. 32, n° 6, Juin 1997.

[MCR95] P. C. Maulik, L. R. Carley, R. A. Rutenbar, *Integer programming based topology selection of cell level analog circuits*, 1995 IEEE, pp 401-412.

[MeG01] : *Mentor Graphics'Analog Mixed-Signal*, http://www.mentorgraphics.com/ams.

[Mey93] R.G. Meyer, *Analysis and design of analogue integrated circuits*, J. Wiley & Sons, 1993.

[Mil97] : N. Milet_Lewis, *Contribution à la modélisation comportementale des circuits analogiques*, thèse Electronique, Bordeaux, 1997.

[MMF01] N. Milet-Lewis, G. Monnerie, A. Fakhfakh, D. Geoffroy, Y. Hervé, H. Lévi, J.J. Charlot, *A VHDL-AMS library of RF blocks models*, BMAS'01, Santa Rosa, octobre 2001.

[Oud00] : J. Oudinot, *Méthodologie de conception d'ASIC mixtes avec VHDL-AMS*, thèse en Electronique et Communications, ENST-Paris, 2000.

[PaM88] S. K. Park, K. W. Miller, *Random number generators: good ones are hard to find*, comm. of the ACM, Vol 31, pp 1192-1201, octobre 1988.

[Pug95] K. Puglia, *Phase-locked loop noise models*, Microwave Engineering Europe, décembre/janvier 1995.

[Raz95] B. Razavi, *Analysis, modeling and simulation of phase noise in monolithic voltage-controlled oscillators*. IEEE 1995 custom integrated circuits conference.

[Raz96] B. Razavi, *A study of phase noise in CMOS oscillators*, IEEE journal of solid-state circuits, Vol. 31, N° 3, mars 1996.

[RCG96] P. Reynaert, L. Callewaert et Al., *ADMIRE : advanced mixed signal design environment*, ICECS 1996, pp 428-431.

[RKL01] : M. Robert, A. Kuntz, L. Le Toumelin, J. F. Pollet, B. Courtois, C. Tordo, *Compact*, le journal de l'industrie française des composants actifs, N° 15, Avril 2001, Ed. SITELESC.

[RSG96] C. J. Richard Shi, N. J. Godambe, *Behavioural fault modeling and simulation of phase-locked loops using a VHDL-A like language*, 1996.

[SnV90] J. G. Sneep, C. J. M. Verhoeven, *A new low-noise 100-MHz balanced relaxation oscillator*. IEEE journal of solid-state circuits, Vol. 25, N° 3, Juin 1990.

[SPL97] K. Sun, H. J. Park, Y. J. Lim, M. Soma, *A top-down design of analog circuits using multi class classifiers as a topology selector*, 5th international conference on VLSI and CAD, pp 232-234, Seoul-Korea, octobre 1997.

[SRC94] R. A. Saleh, D. L. Rhodes, E. Christen, B. A. A. Antao, *Analog hardware description languages*, IEEE 1994 custom integrated circuits conference, pp 15.1.1-15.1.8.

[SSB98] M. Shojaei, M. Sharif-Bakhtiar, *Automatic design of analog circuits based on a behavioral model*, 1998 IEEE, pp 145-148.

[Str67] R. L. Stratonovich, *Topics in the theory of random noise*, New York: Gordon and Breach, 1967.

[TOK99] M. Takahashi, K. Ogawa, K. S. Kundert, *VCO jitter simulation and its comparison with measurement*, Proceeding of the ASP-DAC'99 Asia and South Pacific Design Automation Conference, Vol. 1, pp 85-88, 1999.

[Ver97] VerilogA, *Reference manual*, Cadence, 1997.

[VHD99] VHDL-AMS, *Reference manual*, Mentor Graphics, 1999.

[VLV95] P. Veselinovic, D. Leenaerts, W. Van. Bokhoven et Al, *A flexible topology selection program as part of an analog synthesis system*, IEEE 1995, pp 119-123.

[Wol91] D. H. Wolavar, *Phase-Locked Loop circuit design*, Prentice-Hall, 1991.

[YFW96] H. Z. Yang, C. Z. Fan, H. Wang, R. S. Liu, *Simulated annealing algorithm with multi-molecule: an approach to analog synthesis*, 1996 IEEE, pp 571-575.

[ZMF99] T. Zimmer, N. Milet-Lewis, A. Fakhfakh, B. Ardouin, H. Levi, J.B. Duluc, P. Fouillat, *Hierarchical analogue design and behavioural modelling*, procceding IEEE MSE'99, computer society international conference on microelectronic systems education, Arlington, VA, juin 1999.

Annexe 1

Détermination des paramètres du synthétiseur de fréquence fractionnaire

a) Détermination de la dimension T de l'accumulateur

Pour déterminer la dimension de l'accumulateur T, nous allons considérer le cas général où l'accumulateur attaque un diviseur N/M (N<M).

L'expression du facteur de division non entier est donnée 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 :

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

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

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

On en déduit alors F_{STEP} :

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

D'où l'expression de la fréquence de division T :

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

Selon les valeurs des paramètres du synthétiseur fractionnaire fixées dans le chapitre 3, l'application numérique nous donne :

$$T = \frac{100/6}{5/6} = 20$$

La fréquence de division T est donc égale à 20.

b) Détermination du rapport de division N/N+1

L'expression donnant le rapport de division N_{FRAC} est la suivante :

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

Dans le cas où M = N+1, et en faisant varier K entre 1 et T, on obtient :

$$\frac{1920/6}{100/6} \leq N_{FRAC} \leq \frac{1980/6}{100/6} \quad \Leftrightarrow \quad 19.2 \leq N_{FRAC} \leq 19.8$$

Compte tenu des bornes de N_{FRAC}, ce dernier peut être obtenu en prenant un rapport de division N/N+1 égal à 19/20.

Pour une fréquence à synthétiser égale à 1950 MHz par exemple, calculons l'entrée fractionnaire K à prendre :

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

d'où $K = 10$

c) Détermination des paramètres du filtre

Le filtre de boucle utilisé est celui de la figure 3.9. Pour calculer les valeurs des capacités C_1 et C_2 et de la résistance R, nous appliquons la méthode proposée par Ken Holladay [Hol00]. Il faut alors définir les paramètres suivants :

- La bande de fréquence : **$[F_{maxVCO}, F_{minVCO}]$**
- La largeur du canal : **l_{canal}**
- Le saut fréquentiel à l'intérieur du canal
- La bande passante de la PLL : **BP_{PLL}**
- La sensibilité du VCO : **K_{VCO}**
- Le courant de la pompe de charge : **I_{pompe}**

La méthode se résume ensuite en sept étapes :

1. Calcul de F_{step} : $F_{step} = F_{maxVCO} - F_{minVCO}$

2. Calcul de N : $N = \frac{F_{maxVCO}}{l_{canal}}$

3. Calcul de la fréquence naturel F_N $\frac{2.BP_{PLL}}{2\pi\left(\xi + \frac{1}{4\xi}\right)}$

ξ étant le facteur d'amortissement, typiquement égal à 0.707.

4. Calcul de C_2 : $C_2 = \frac{I_{CP}.K_{VCO}}{N(2\pi F_N)^2}$

5. Calcul de R_1 : $$R_1 = 2\xi\sqrt{\frac{N}{I_{CP}K_{VCO}C_2}}$$

6. Calcul de C_1 : $$C_1 = \frac{C_2}{10}$$

7. Calcul du temps d'établissement T_S : $$T_S = \frac{-\left(\ln\frac{F_A}{F_{STEP}}\right)}{F_N.2\pi.2\xi}$$

F_A est une fréquence qui est généralement prise égale à 1000 Hz.

Les caractéristiques de notre synthétiseur de fréquence sont les suivantes :

- F_{maxVCO} = 350 MHz et F_{minVCO} = 300 MHz
- La largeur du canal l_{canal} = 5/6 MHz
- Le saut fréquentiel à l'intérieur du canal = 5/6 MHz
- La bande passante de la PLL BP_{PLL} = 150 KHz
- La sensibilité du VCO K_{VCO} = 50 MHz/V
- Le courant de la pompe de charge I_{CP} = 100 μA

En faisant une application numérique, on obtient les valeurs suivantes pour R1, C1 et C2 :

$R_1 = 33.6K\Omega$, $C_1 = 15pF$ et $C_2 = 0.15nF$

Annexe 2

Modèle VHDL-AMS de l'oscillateur synchrone 2.4 GHz de type COLPITTS

```
library disciplines;
use disciplines.electromagnetic_system.all;
library IEEE;
use IEEE.math_real.all;

entity oscillateur_synchrone is
    generic(vdd : real := 3.0;
            L : real := 3.252e-9;
            c1 : real := 1.7e-12;
            c2 : real := 3.639e-12;
            cu : real := 441.0e-15;
            gm : real := 40.0e-3;
            gpi : real := 5.5e-3;
            gc : real := 6.54e-3;
            io : real := 2.0e-3;        -- amplitude du courant de synchronisation --
            vo : real := 215.0e-3;          -- amplitude du signal de synchronisation --
            vs : real := 540.0e-3;          -- amplitude du signal de sortie
            duree : real := 500.0e-12;      -- durée de l'impulsion --
            vtran : real := 0.0;
```

```
            Abruit : real := 0.0013;
            fm : real := 600.0e3;              -- fréquence d'offset --
            tdel : real := 0.0;
            trise : real :=4.0e-15;
            tfall : real :=4.0e-15);
    port (terminal inp, inm, outp, outm : electrical);
end entity oscillateur_synchrone;

                                    -- oscillateur COLPITTS --

architecture comportement of oscillateur_synchrone is
    quantity vin across iin through inp to inm;
    quantity vout across iout through outp to outm;
    quantity wo : real;
    quantity fo : real;
    quantity cg : real;
    quantity cb : real;
    quantity cw : real;
    quantity i : real;
    quantity marge : real;
    quantity finf : real := 4.0e8;
    quantity fsup : real := 4.0e8;
    quantity fi : real := 4.0e8;
    quantity wi : real;
    quantity a : real := 0.2;
    quantity ts : real;
    quantity ts_bruite : real;
    quantity compteur : real := 0.0;
    quantity t : real;
    signal n : real := 1.0;
    signal y : real := 6.0;
    signal periode : real := 2.5e-9;
    signal t2 : real := 0.0;
    signal fs : real := 2.4e9;
    signal compt : real := 1.0;
    signal bruit : real := 0.0;
    signal jitter : real := 0.0;
    signal compteur_n1 : real := 2.085e-10;
    signal t0 : real := 0.0;
    signal temoin : real := -1.0;
```

```
    signal vss : real := vdd;
    signal M : real := 1.0;
    signal somme_DTn2 : real := 0.0;
    signal somme : real := 0.0;
    signal DTn : real := 0.0;
    signal DTn2 : real := 0.0;
    signal cycle_jit : real := 0.0;
    signal DTn1_n : real := 0.0;
    signal cycle_to_cycle_jit : real := 0.0;
    signal absolute_jit : real := 0.0;
    signal carre : real := 0.0;
    signal Lphi : real := 0.0;

begin
iin == 0.0;
                    -- détermination des caractéristiques d'entrée --
                        -- calcul de la période d'entrée --
calcul_periode : process
begin
    wait until vin'above(vtran) = true;
    if compt > 1.0 then
            t2 <= t;
            periode <= t - t2;
    else
            t2 <= t;
            periode <= 2.5e-9;
    end if;
    compt <= compt +1.0;
end process calcul_periode;

                    -- calcul de la fréquence et de la pulsation d'entrée --
fi * periode == 1.0;
wi == MATH_2_PI * fi;

                        -- calcul des paramètres de sortie --

                    -- calcul de la fréquence et de la pulsation propres --
    wo == sqrt(((gm+gpi)*(L*gc/(c1+c2))+1.0)*(1.0/L)*(1.0/(cu+((c1*c2)/(c1+c2)))));
    fo == 0.5*wo/MATH_PI;
```

```
                    -- détermination de l'harmonique de synchronisation --
sub_harmonic : process
begin
    wait until vin'above(vtran) = true;
    n <= periode * fo;
    if n <= 0.5 then y <= 1.0;
    else y <= round(n);
    end if;
end process sub-harmonic;

            -- calcul de la nème harmonique du courant de synchronisation --
    a == duree*fi;
    i * (a*y*y*MATH_PI*MATH_PI) == 2.0*io*(1.0-cos(a*y*MATH_PI));

                        -- calcul de la plage de synchronisation --

                        -- calcul des facteurs de compliance --
    cg == (1.0/(8.0*MATH_PI*MATH_PI))*(1.0/fo)*(gc/(cu*(c1+c2)+c1*c2));
    cb == (1.0/(4.0*MATH_PI))*(1.0/((c1+c2)*(c1+c2)))*(L*gc*(gm+gpi)/
          ((1.0+L*gc*(gm+gpi)/(c1+c2)))+c1*c1/(cu+c1*c2/(c1+c2)));

                        -- calcul de la marge de capture --
    cw == sqrt(cg*cg+cb*cb);
    marge == 2.0*(i/vo)*cw;

                        -- détermination de la fréquence de sortie --
    finf * y == (fo-0.5*marge);
    fsup * y == (fo+0.5*marge);
synchro : process
begin
    wait until compteur'above(ts_bruite) = true;
    if fi > finf and fi < fsup then
            fs <= fi * y;
            jitter <= 0.0;
    else    fs <= fo;
            jitter <= bruit;
    end if;
end process synchro;
                        -- calcul de la période de sortie --
ts * fs == 1.0;
```

```
-- modélisation du bruit de phase --

-- génération d'un signal aléatoire à l'aide de la fonction UNIFORM --
signal_aleatoire : process
    variable seed1 : positive := 19823;
    variable seed2 : positive := 124;
    variable x : real;
begin
    wait until compteur'above(ts_bruite) = true;
    UNIFORM(seed1, seed2, x);
    bruit <= Abruit*2.0*(x - 0.5);
end process signal_aleatoire;

-- calcul de la demi période bruitée --
ts_bruite == 0.5*ts*(1.0+jitter);

-- calcul des caractéristiques du bruit de phase --
t == now;
compteur == t - t0;
compteur2 == t - t02;
-- calcul de la gigue cyclique --
cycle_jitter : process
begin
    wait until compteur'above(ts_bruite) = true;
    DTn <= compteur -  0.5 * ts;
    DTn2 <= DTn**2.0;
    somme_DTn2 <= somme_DTn2 + DTn2;
    cycle_jit <= sqrt(somme_DTn2/M);
    M <= M + 1.0;
    t0 <= t;
end process cycle_jitter ;

-- calcul de la gigue cycle-à-cycle --
cycle_to_cycle_jitter : process
begin
    wait until compteur'above(ts_bruite) = true;
    DTn1_n <= compteur - compteur_n1;
    carre <= DTn1_n**2.0;
    somme <= somme + carre;
    cycle_to_cycle_jit <= sqrt(somme/M);
```

```
    compteur_n1 <= compteur;
    M <= M + 1.0;
    t0 <= t;
end process cycle_to_cycle_jitter ;

                                    -- calcul de la gigue absolue --
absolute_jitter : process
begin
    wait until compteur'above(ts_bruite) = true;
    DTn <= compteur -  0.5 * ts;
    absolute_jit <= absolute_jit + DTn;
    t0 <= t;
end process cycle_jitter ;

                    -- calcul de la densité spectrale de puissance unilatérale --
power_density : process
begin
    wait until compteur'above(ts_bruite) = true;
    if cycle_jit > 0.0 then
    Lphi <= 10.0*log10(cycle_jit * cycle_jit * fs*fs*fs/(2.0*fm*fm));
    else Lphi <= 0.0;
    end if;
    t0 <= t;
    temoin <= -temoin;
end process power_density ;

                                    -- génération du signal de sortie --
vss <= vdd - 0.5*(temoin-1.0)*(-0.5*vs) + 0.5*(temoin+1.0)*0.5*vs;
vout == vss'ramp(tdel,trise,tfall);
end architecture comportement;
```

Résumé

Avec les nouvelles tendances que sont les System-on-Chip (SoC) et les ASIC mixtes, les outils de CAO analogiques doivent évoluer pour permettre une conception hiérarchique, basée sur la ré-utilisation de blocs, utilisant largement la modélisation comportementale des circuits. L'avènement des langages de description standards pour les circuits analogiques et mixtes (VHDL-AMS par exemple) ouvre la voie à l'amélioration attendue mais il reste à développer divers outils et méthodes.
Les travaux présentés dans ce mémoire apportent une contribution à la modélisation comportementale des circuits analogiques et mixtes en termes de méthodologie et de développement de bibliothèques de modèles standards. Ainsi, nous proposons une méthode systématique de modélisation comportementale, une bibliothèque de modèles pour les circuits RF et une étude du bruit de phase aboutissant à une modélisation de ce phénomène dans les oscillateurs.

Abstract

The new tendencies in electronic system design are the development of System-on-Chip (SoC) and mixed ASIC. The CAD tools should evolve to allow a hierarchical design, based on the re-use of blocks and the behavioral modeling of circuits. The development of standard description languages for analogue and mixed systems (VHDL-AMS for example) opens the way to the expected improvement, but different methods and tools should be developed.
This work presents a contribution to the behavioral modeling of analogue and mixed circuits in terms of methodology and development of standard model libraries. We propose a systematic method of behavioral modeling, a library of models for RF circuits and a study of the phase noise effect in oscillators ending in the behavioral modeling of this phenomenon.

Mots-clés

Conception Assistée par Ordinateur (CAO)
Circuits analogiques et mixtes
Circuits Radio-Fréquence (RF)
Bruit de phase des oscillateurs
Modélisation comportementale
Langage de description matérielle (VHDL-AMS, VerilogA)

Key words

Computer Aided Design (CAD)
Analogue and Mixed Circuits
Radio-Frequency Circuits
Oscillator phase noise
Behavioral modeling
Hardware Description Language (VHDL-AMS, VerilogA)