UNIVERSITE DE LIMOGES

ECOLE DOCTORALE SCIENCES - TECHNOLOGIE - SANTE

FACULTE DES SCIENCES ET TECHNIQUES

XLIM – Département C²S²

Année : 2006 Thèse n°13-2006

Thèse

pour obtenir le grade de

Docteur de l'Université de Limoges

Discipline : Electronique des Hautes Fréquences et Optoélectronique

Spécialité : "Communications Optiques et Microondes"

Présentée et soutenue par

Abderrazak BENNADJI

Le 14 avril 2006

Implémentation de modèles comportementaux d'amplificateurs de puissance dans des environnements de simulation système et co-simulation circuit système

Thèse dirigée par Edouard NGOYA

JURY :

Madame Geneviève Baudoin	Professeur, ESIEE, Marne la Vallée	Rapporteur
Monsieur Jean-François Diouris	Professeur, Université de Nantes	Rapporteur
Monsieur Edouard Ngoya	Directeur de Recherche CNRS - IRCOM	Examinateur
Monsieur Raymond Quéré	Professeur à l'Université de Limoges	Examinateur
Monsieur Jean-Michel Nébus	Professeur à l'Université de Limoges	Examinateur
Monsieur Ramine Nikoukhah	Directeur de Recherche, INRIA, Rocquencourt	Examinateur
Monsieur Bernard Huyart	Professeur, ENST, Paris	Invité
Monsieur Alain Mallet	Ingénieur, CNES, Toulouse	Invité
Monsieur Laurent Constantias	Ingénieur, CELAR,Bruz	Invité
Monsieur Jean Rousset	Ingénieur, CNRS, Université de Limoges	Invité

A ma mère, à mon père,

A mon épouse et toute ma famille,

A tous ceux qui me sont chers,…

REMERCIEMENTS

Ce travail a été effectué à l'Institut XLIM, UMR CNRS n° 6172.

Je remercie Monsieur le Professeur P.Y. GUILLON, Directeur de XLIM de m'avoir accueilli dans ce laboratoire.

Je remercie ma tendre épouse Faiza pour son soutien et ses encouragements dans les instants les plus difficiles. Merci pour son aide et pour avoir supporté mon humeur durant cette période dense en activité, infiniment merci.

J'exprime mes remerciements à Monsieur R. QUERE, Professeur à l'université de Limoges, de m'avoir accueilli au sein de l'équipe « Circuits et sous-ensembles électroniques non linéaires ».

Je remercie mon directeur de thèse Monsieur E. NGOYA, directeur de recherche au CNRS pour tout ce temps passé ensemble et pour cette formidable disponibilité. Merci de m'avoir transmis en partie la rigueur qui le caractérise. Je lui suis tout particulièrement reconnaissant de m'avoir permis de découvrir beaucoup d'aspects de la modélisation des amplificateurs que j'ignorais.

Ce travail n'aurait jamais pu aboutir sans Arnaud Soury, qui a toujours su me consacrer son temps, me guider et me conseiller dans l'implémentation numérique des modèles.

Je tiens également à exprimer mes sincères remerciements à Madame Geneviève Baudoin, Professeur à l'université de Noisy le Grand ainsi qu'à Monsieur Jean-François Diouris, Professeur à l'université de Nantes pour l'intérêt qu'ils ont porté à ces travaux en acceptant de les rapporter.

Je témoigne ma reconnaissance à Monsieur Ramine Nikoukhah (Directeur de Recherche à l'INRIA), Monsieur A. MALLET (Ingénieur au CNES), Monsieur Bernard Huyart (Professeur à ENST de Paris), Monsieur Laurent Constantias (Ingénieur à CELAR), Monsieur Jean Rousset (Ingénieur au CNRS) d'avoir accepté de participer à ce jury.

J'adresse ma vive reconnaissance à Monsieur J.M. NEBUS, Professeur à l'Université de Limoges, qui a su me transmettre sa passion au cours des années de formation passées à la faculté des sciences de Limoges.

Je ne terminerai pas mes remerciements sans avoir une pensée sympathique pour tous les thésards, enseignants-chercheurs et toutes les personnes que j'ai côtoyé au cours de ces trois années, et notamment les membres de l'équipe C2S2 qui savent si bien rendre agréable le cadre de travail, et tout particulièrement : Sébastien MONS (Chargé de recherche CNRS), Christophe MAZIERE, Tibault REVEYRAND, Tony GASSELING, Stéphane DELLIER, Christophe CHARBONNIAUD, Alan LAYEC, François MACRAIGNE, Mohamed EL YAAGOUBI**,** François-Xavier ESTAGERIE, Zhour MADINI, Grégoire NANFACK-NKONDEM, Audrey PHILIPPON, Floria BLANCHET, Hind BOUSBIA, Zoheir SASSI, Bo WANG, Virginie BERNICAL, Vincent ARMENGAUD.

Enfin je tiens également à faire part de ma gratitude à Marie Claude pour sa gentillesse et son professionnalisme par qui j'ai pu concentrer toute mon attention sur ce travail.

TABLE DES MATIERES

Introduction Générale

Introduction Générale

Les systèmes de télécommunications actuels tirent pleinement profit des technologies monolithiques ces dernières années. La conception de tels systèmes nécessite des outils d'analyse assistés par ordinateur « **CAO** » performants de par le fait que, d'une part, l'espace d'intégration est fortement réduit, ce qui augmente les interactions entre composants, et que, d'autre part, les ajustements post-réalisation sont quasi-impossibles.

La simulation et la modélisation de tels systèmes est un sujet de recherche majeur depuis plusieurs années. La complexité structurelle de ces systèmes et la disparité des signaux à transmettre sont tels qu'ils représentent des obstacles importants à l'évaluation des performances. La prédiction de ces performances devient de plus en plus critique en vue d'une réalisation optimale en termes de dimensionnement, de fiabilité et de coût. Cela nécessite des simulations de type système qui soient performantes. C'est dans ce contexte que des efforts importants ont été conduits ces dernières années afin d'établir des techniques de macro-modélisation de circuits fonctionnels (amplificateurs, mélangeurs, oscillateur, etc.) alliant précision, stabilité et rapidité. Aujourd'hui ces travaux ont atteint une certaine maturité, cependant ils se sont à ce jour essentiellement attachés aux développements de formalismes de modélisation et les modèles qui en découlent ne sont pas encore disponibles dans les environnements de simulation système. Pour que les concepteurs de circuits et de systèmes puissent pleinement tirer profit de ces nouvelles techniques, il est nécessaire d'étudier l'implémentation de ces modèles dans les environnements de simulation système généralistes. Ce travail constitue le point d'ancrage de cette thèse dont l'objectif a été de réaliser la synthèse des modèles comportementaux d'amplificateurs récemment mis au point, notamment à l'Université de Limoges et d'étudier leur implémentation dans les simulateurs système.

L'implémentation de macro-modèles dans un environnement système nécessite une attention particulière sur le coût de calcul et la précision ainsi que sur la stabilité du système d'équations. Pour cela un élément essentiel de cette thèse a été l'étude des techniques numériques permettant d'optimiser le compromis entre la précision et les coûts de calcul.

Le mémoire de thèse est organisé en trois chapitres. Dans le premier chapitre nous présenterons le contexte lié à l'environnement de simulation système. Pour cela nous ferons un rappel sur les principales méthodes de simulation des circuits non linéaires et le principe de la simulation système. Nous présenterons dans la deuxième partie de ce chapitre une synthèse des travaux qui ont été menés pour la prise en compte des effets dispersifs dans les modèles

comportementaux d'amplificateurs. Nous nous appesantirons plus particulièrement sur les modèles de Volterra développés au sein de notre équipe de recherche.

Le deuxième chapitre est divisé en deux parties. La première partie est consacrée à la description des simulateurs que nous avons utilisé dans le cadre de notre travail. Il s'agit des simulateurs Scilab/Scicos et Matlab/Simulink. La deuxième partie du chapitre présente les approches mathématiques que nous avons adopté pour implémenter les modèles de Volterra dans les environnements de simulation système généraux. Ces approches nous permettent d'éliminer le calcul de l'intégrale de convolution, qui est une procédure très lourde et d'exprimer les réponses des dispositifs sous la forme d'équations différentielles. Des exemples de simulation avec des signaux élémentaires et des signaux de communication numérique y sont présentés qui permettent de valider la démarche suivie, tant pour ce qui concernent la précision que le temps de calcul du modèle.

Le troisième chapitre présente une interface de co-simulation système-circuit permettant au simulateur de système d'accéder à un simulateur de circuit pour chaque échantillon du signal. La co-simulation est un outil d'échange essentiel entre les domaines système et circuit qui est aujourd'hui encore un chaînon manquant dans l'électronique analogique RF et micro-onde. La co-simulation permettrait d'une part, au concepteur de systèmes d'accéder le cas échéant à la plus grande finesse possible de description du circuit, et d'autre part au concepteur de circuit la possibilité de valider ses réalisations avec les standards de communications disponibles dans le simulateur système. L'interface de co-simulation que nous avons étudié est basée sur le mécanisme de communication inter-processus pour les applications Windows. La démarche décrite dans ce chapitre est illustrée par un exemple de co-simulation entre Matlab/Simulink™, Scilab/Scicos™ et le simulateur circuit Xpedion/Golden Gate™.

Enfin, la conclusion présente le résumé des différents résultats obtenus dans cette thèse.

CHAPITRE I

Simulation système et modélisation comportementale des amplificateurs de puissance

I. Introduction

La simulation des systèmes de télécommunication implique une démarche hiérarchique descendante dite « top-down ». Celle-ci qui décompose le problème de la simulation d'un système complexe en une suite de problèmes élémentaires où nous profitons d'algorithmes de simulation spécifique à chaque niveau étudié. On peut dégager trois familles d'outils de simulation (**système - circuit - composant**), comme illustre la figure I-1.

Si l'analyse dite de type « circuit » et « composant » a beaucoup progressé, l'analyse « système » souffre encore d'un manque important de finesse sur la description des phénomènes mis en jeu dans les circuits micro-ondes non linéaires. De plus, les simulateurs système actuels souffrent d'un manque de modèles comportementaux évolués capables de décrire avec assez de précision les dispersions et les effets non linéaires dynamiques engendrés par les équipements d'émission.

Dans ce chapitre, nous présentons dans un premier temps les principales méthodes de simulation des circuits non linéaires, en mettant l'accent sur leurs points forts et leurs limitations. Puis nous présentons le principe de la simulation système. Dans la dernière partie de ce chapitre, nous présentons les travaux qui ont été menés ces dernières années sur les modèles comportementaux d'amplificateurs, nous présentons un état des lieux des principaux modèles comportementaux d'amplificateurs relevés dans la littérature en mettant l'accent sur leurs points forts et leurs limitations.

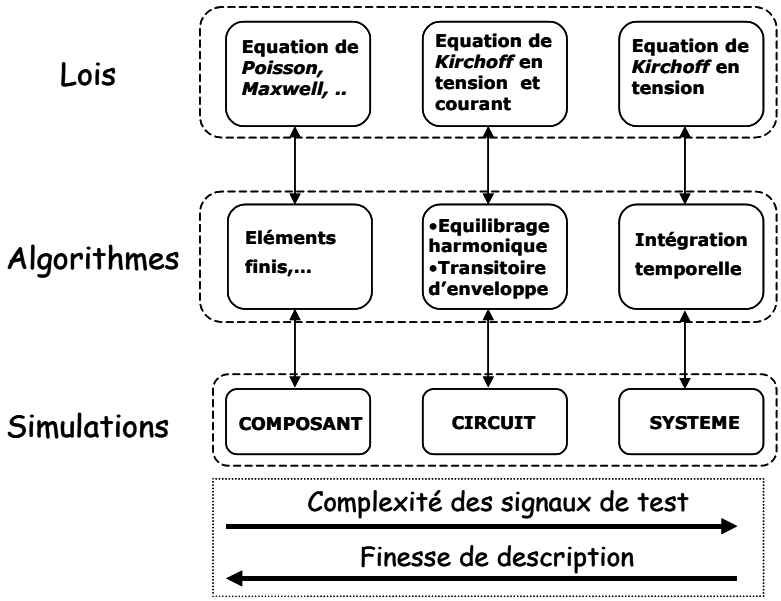


Figure I-1 : Synoptique des outils de simulation

II. Les simulateurs circuit

Nous présentons dans cette section un panorama des principales méthodes utilisées pour la simulation de circuits électriques. Les méthodes présentées ici peuvent être regroupées dans trois grandes familles, celle basée sur l'intégration temporelle, celle basée sur l'équilibrage harmonique et la méthode de transitoire d'enveloppe.

Dans une première partie, nous présentons la méthode d'intégration temporelle directe. Dans la deuxième partie, nous présentons la méthode d'Equilibrage Harmonique et dans la troisième partie la méthode de transitoire d'enveloppe.

Le point de départ pour ces méthodes est le système d'équations différentielles algébrique ordinaire provenant de l'analyse nodale modifiée de la forme :

$$\int_0^t f(\tau).x(t-\tau)d\tau + \frac{d}{dt}q(x(t)) + i(x(t)) + u(t) = 0 \qquad \text{(I-1)}$$

où $q(t) \in \mathbb{R}^N$, $i(t) \in \mathbb{R}^N$ et $u(t) \in \mathbb{R}^N$ sont respectivement les équivalents « nodal modifié » des charges et des flux des éléments réactifs, des courants convectifs et des générateurs d'excitation.

II.1. LA TECHNIQUE D'INTEGRATION TEMPORELLE DIRECTE

La simulation d'un circuit en utilisant la méthode d'intégration temporelle se résume à résoudre numériquement un système d'équation non linéaire de type de l'équation (I-1). Pour ce faire, nous passons du circuit physique à un système d'équations par une méthode de formulation des équations, en représentant chaque composant de base (résistance, condensateur, inductance, etc.) par son modèle mathématique et en appliquant les contraintes de connexions imposées par les lois de Kirchhoff.

La méthode d'Intégration Temporelle Directe (I.T.D) [I-1] est la méthode la plus naturelle pour étudier la réponse d'un circuit, elle permet l'analyse de tous les circuits électriques quel que soit le type d'excitation (périodiques, apériodique, impulsionnel, etc …). Pour ce faire, l'axe de temps de l'équation (I-1) est discrétisé et les dérivées sont approximées à l'aide des formules LMS (Linear Multistep) telles que les formules d'Euler ou de Gear [I-2].

A titre d'exemple le courant traversant une capacité (Figure I-2) a la forme suivante :

$$i(t) = C.\frac{dv(t)}{dt} \tag{I-2}$$

La représentation du courant (I-2) à l'instant $t = t_n$ en utilisant la formule de Trapèze conduit à la relation suivante :

$$i_n = \frac{2C}{h_n}.v_n - \left[i_{n-1} + \frac{2C}{h_n} v_{n-1} \right] \tag{I-3}$$

Nous constatons, que le courant se décompose en deux termes, le premier dépendant de l'instant courant et simulant une conductance linéaire de valeur $\frac{2C}{h_n}$, et le deuxième dépend de l'instant précédant et simule une source de courant ayant pour expression $i_{n-1} + \frac{2C}{h_n} v_{n-1}$.

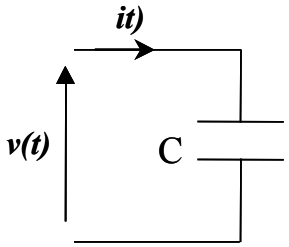


Figure I-2 : Capacité linéaire

La méthode d'intégration temporelle permet de considérer n'importe quel type d'excitation (périodique, impulsionnelle, ….) et son application se trouve dans l'analyse des circuits à transitoire rapide. En effet comme nous pouvons en déduire, l'efficacité de cette méthode dépend du pas d'intégration choisi. Un pas d'intégration petit permet d'avoir une bonne précision mais rendra le coût de calcul plus important, alors qu'un pas important rendra la précision moins bonne mais améliorera le coût de calcul. Cependant le fait que l'intégration temporelle directe doit toujours passer par le régime transitoire avant d'atteindre le régime établi peut être un inconvénient lorsque l'on ne s'intéresse qu'à ce dernier. Le problème de précision peut être partiellement résolu en utilisant un pas d'intégration variable [I-3], et le deuxième problème du comportement en état établi peut être résolu en utilisant les méthodes balistiques [I-4][I-5] pour passer directement au régime établi sans passer par le régime transitoire.

Malgré cela, les méthodes temporelles présentent des difficultés difficilement contournables de prise en compte des éléments distribués dans la simulation des circuits micro-ondes. Pour cette raison on leur préfère les méthodes tempo-fréquentielles permettant de calculer directement le régime établi du circuit.

II.2. L'ÉQUILIBRAGE HARMONIQUE

L'utilisation des méthodes d'intégration temporelle pour résoudre les équations des circuits électroniques qui contiennent des éléments distribués, faisant intervenir des produits de convolution sur des durées de mémoire importantes par rapport aux variations du signal hyper fréquence, conduit à un temps de calcul prohibitif. Pour résoudre ce problème, la méthode de l'équilibrage harmonique a été introduite dans les années 60 par Baily et Lindenlaub [I-6] [I-7] [I-8] [I-9].

Cette méthode est une technique puissante pour l'analyse des circuits non-linéaires à haute fréquence tels que les amplificateurs de puissance, les mélangeurs, et les oscillateurs. La méthode reprise au début des années 80 par beaucoup d'auteurs est devenue rapidement dans les années 90 la méthode de référence de simulation des circuits hyperfréquences MMIC. Plus récemment, avec les progrès en analyse numérique autour des méthodes itératives de sous espaces de Krylov [I-10], le champ d'application de l'équilibrage harmonique a été étendu aux circuits à grande intégration comme les circuits intégrés RF (RFIC).

Le principe de base de l'équilibrage harmonique consiste à décomposer le système en sous ensemble linéaires et non linéaires (Figure I-3), de telle sorte que nous pouvons résumer ce processus par les étapes suivantes :

- Division du système en deux sous-ensembles :
    - Sous-ensemble linéaire regroupant tous les éléments linéaires définis dans le domaine fréquentiel ;
    - Sous-ensemble non linéaire regroupant tous les éléments définis par des fonctions mathématiques non linéaires.
- Utilisation d'une interface de transformation de Fourier qui permet de relier les deux sous-ensembles.
- Résolution du système d'équations non linéaire en utilisant les méthodes de Newton-Raphson et Krylov.

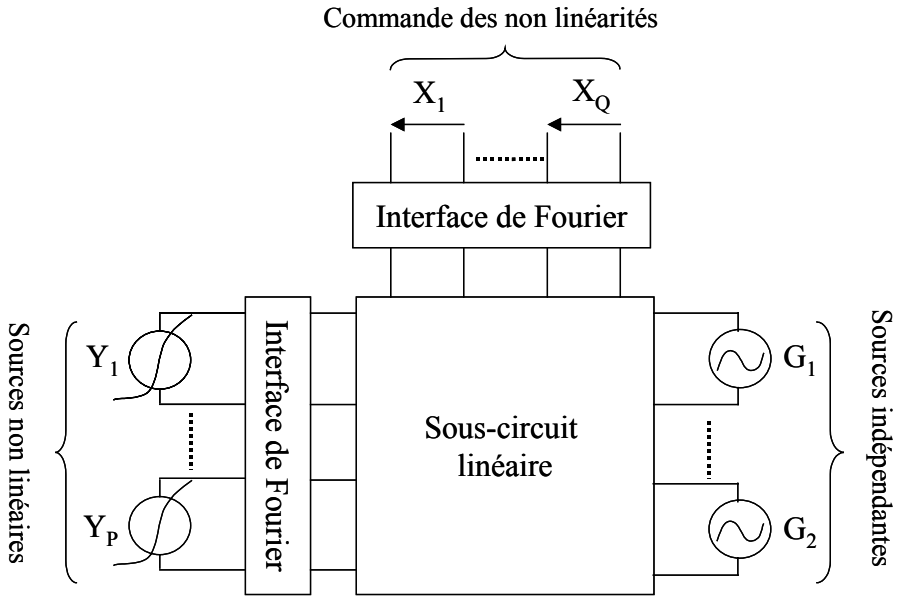


Figure I-3 : Synoptique d'équilibrage harmonique

A partir du schéma I-3, il est possible de définir l'équation d'équilibrage harmonique en considérant les composantes des commandes comme étant les sorties du circuit alors que les générateurs G et Y sont les entrées, en effet en utilisant le théorème de la superposition et les matrices de transfert du circuit, nous avons l'équation donnée ci-dessous.

$$F\left(\hat{X}_k\right) = \hat{X}_k - \hat{A}_k.\hat{Y}_k\left(\hat{X}\right) - \hat{B}_k.\hat{G}_k = 0 \qquad \text{(I-4)}$$

Avec k est l'indice de la composante fréquentielle, $\hat{X}_k$ est le vecteur des kièmes composantes fréquentielles des commandes, $\hat{A}_k$ est la matrice de transfert des non-linéarités vers les commandes et $\hat{B}_k$ est la matrice caractérisant le transfert des sources indépendantes vers les commandes.

Les paramètres fondamentaux dans l'analyse par équilibrage harmonique sont les fréquences fondamentales et les nombres d'harmoniques suivant chaque fréquence. Le nombre de fréquences fondamentales est pratiquement limité à trois, au-delà les performances du simulateur deviennent trop faibles. Le nombre d'harmoniques est un paramètre critique laissé à la charge de l'utilisateur. Il nécessite un compromis difficile entre précision de calcul des non-linéarité et de validité des modèles des éléments. Dans la pratique, on réalise l'analyse du circuit en augmentant progressivement les nombres d'harmoniques jusqu'à ce que l'on obtienne une stabilisation du résultat.

II.3. TRANSITOIRE D'ENVELOPPE

Les méthodes d'équilibrage harmonique et d'intégration temporelle ne sont pas adaptées aux signaux modulés car si nous essayons de rester dans le domaine temporel, nous devons échantillonner correctement le signal de plus haute fréquence (porteuse), ce qui conduit à des temps de simulation rédhibitoires ou alors nous analysons les signaux dans le domaine fréquentiel et nous discrétisons le spectre de modulations, ce qui conduit également à un nombre important de raies de fréquences.

La solution apportée pour ces problèmes difficilement solvables est la méthode de transitoire d'enveloppe initiée dans [I-11] [I-12] [I-13]. Le principe de base de cette méthode consiste à dissocier les techniques d'analyse utilisées pour les fréquences porteuses et les signaux modulants. Elle consiste à analyser le comportement d'enveloppe par la méthode temporelle, et de l'équilibrage harmonique autour des fréquences porteuses. De façon générale nous considérons le signal dans sa forme bande de base, c'est à dire constitué d'un ensemble d'enveloppes complexes autour de chaque harmonique de la porteuse, soit :

$$x(t) = \Re e\left(\sum_{k=0}^{N} \tilde{V}_k(t).e^{j.k.\omega_0.t} \right) \tag{I-5}$$

où k représente le numéro d'harmonique de la porteuse, $\tilde{V}_k(t)$ est une grandeur complexe, que l'on décompose en deux termes en quadrature :

$$\tilde{V}_k(t) = \tilde{V}_{ik}(t) + j\tilde{V}_{qk}(t) \tag{I-6}$$

Le principe mis en œuvre consiste à échantillonner l'enveloppe du signal modulé avec un pas approprié à la dynamique de l'enveloppe et de profiter de l'équilibrage harmonique pour l'analyse de chaque échantillon. Ainsi le signal modulé se trouve découpé en une approximation constituée d'une série de tronçons de sinusoïdes qu'il devient aisé de simuler à l'aide de la technique de l'équilibrage harmonique. Ce principe est illustré sur la figure I-4 :

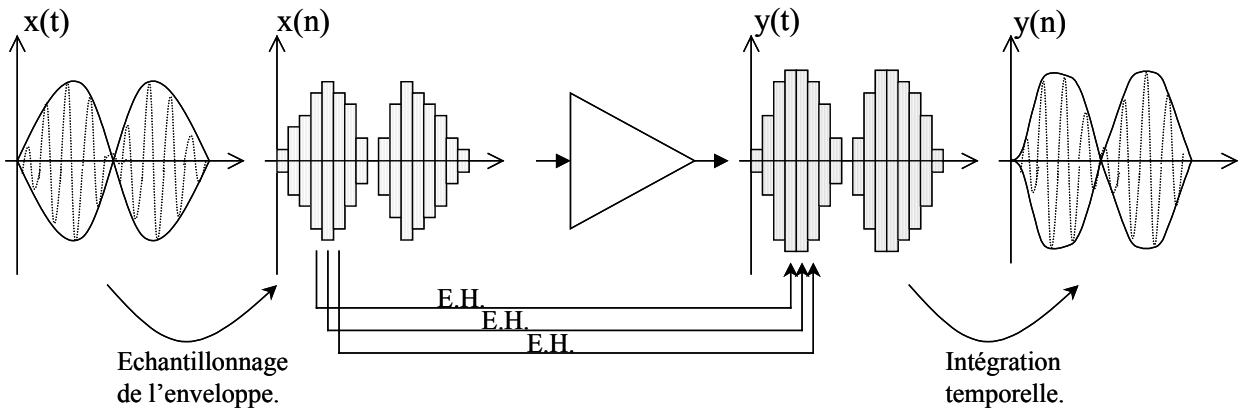


Figure I-4 : Schéma de principe du transitoire d'enveloppe

Pour illustrer ce principe, nous prenons le même exemple du paragraphe (II-1) (Figure I-2). Si nous appliquons aux bornes de la capacité un signal d'excitation complexe $v(t)$ comme montrée dans l'équation (I-5), le courant se met alors sous la forme :

$$i(t) = C.\Re e\left(\sum_{k=0}^{N}\left[\frac{d\tilde{V}_k(t)}{dt} + j.\omega_k.\tilde{V}_k(t)\right].e^{j.\omega_0.t}\right) = \Re e\left(\sum_{k=0}^{N} I_k(t).e^{j.\omega_0.t}\right) \qquad \text{(I-7)}$$

Le courant $I_k(t)$ est composé de deux termes, un terme en régime établi ($j.\omega_k.\tilde{V}_k(t)$) qui va être traité dans le domaine fréquentiel par la méthode d'équilibrage harmonique et le terme transitoire $\frac{d\tilde{V}_k(t)}{dt}$ dans le domaine temporel par la méthode d'intégration temporelle.

II.4. LES OUTILS DE SIMULATION CIRCUIT

Il existe plusieurs simulateurs de circuit sur le marché, nous pouvons les classifier en trois catégories citées ci-dessous.

II.4.1. Les simulateurs de type temporel

Parmi les simulateurs temporels les plus répandus, on peut citer les simulateurs Spice, Spectre S, Hspice et Eldo, ces simulateurs utilisent la méthode d'intégration temporelle directe et le simulateur Spectre RF de la société Cadence qui lui utilise la méthode de shooting.

II.4.2. Les simulateurs de type transitoire d'enveloppe

Trois simulateurs de transitoire d'enveloppe sont aujourd'hui disponibles dans les principales plates-formes industrielles de simulation analogique comme Cadence (GoldenGate-Xpedion), Agilent- Eesof (ADS) et Mentor Graphics (Eldo-RF).

II.4.3. Les simulateurs de type harmonique balance

Depuis deux décennies, la méthode d'équilibrage harmonique a été adoptée dans la quasi-totalité des outils de simulation industriels s'adressant aux circuits RF et micro-ondes. On peut citer les simulateurs suivants : ADS (Agilent), GoldenGate (Xpedion), Microwave office (AWR) et Eldo RF(Mentor Graphics).

III. Les simulateurs système

Depuis quelques années, nous voyons l'émergence d'un grand nombre de plates-formes pour la simulation système plus particulièrement destinées aux applications de traitement de signal. La plus populaire d'entre elles est sans doute le module Simulink de l'environnement MATLAB. Nous trouvons également d'autres ateliers comme Scicos de l'environnement Scilab, Ptolemy, SPW de Cadence, COSSAP de Synopsis, encore LabView de National Instruments.

Chaque simulateur système propose une nomenclature et un formalisme différent. C'est pourquoi on donne dans ce paragraphe les notions principales des modèles utilisés par chaque simulateur.

Un système est défini comme un ensemble interconnecté de blocs. Chaque bloc est composé de N portes d'entrée et M portes de sortie. La simulation doit représenter le flux d'informations temporelles qui circulent dans le système. Pour cela, on suppose l'existence de blocs d'entrée produisant les conditions initiales.

Les nœuds sont interconnectés et l'information transférée entre deux nœuds est nommée *tokens*. Un *token* peut représenter un échantillon réel, une matrice ou toute autre structure de données. La nature des *tokens* acceptés par chaque porte d'un bloc fait partie de la définition du bloc. Dans un système, il est possible d'avoir des *tokens* de différents types. Il n'y a pas a priori une signification physique de l'information traitée par le simulateur.

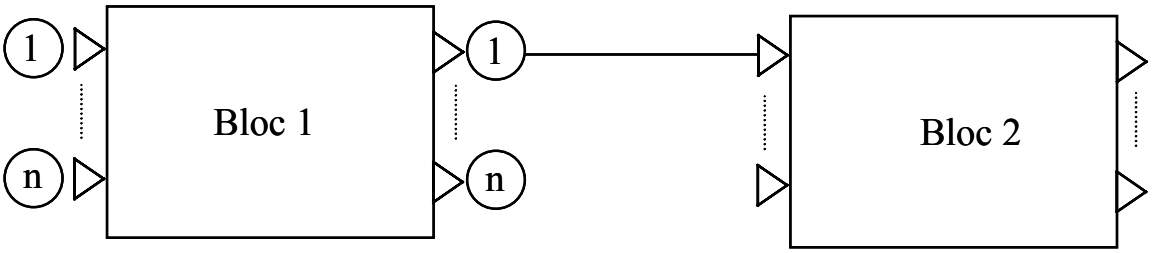


Figure I-5 : Connexion entre deux blocs système

III.1. LES MODELES DE CALCUL

A un niveau abstrait, un système peut être considéré comme un ensemble de composants qui ont des propriétés, et entre lesquels existent des relations. Les différents composants d'un modèle peuvent appartenir à des domaines techniques différents tels que l'électronique analogique, l'électronique numérique, la mécanique, la thermodynamique, l'optique etc. Ces domaines techniques ont des méthodes de modélisation différentes, et ne considèrent donc pas les composants et les relations entre composants de la même façon.

L'ensemble de « lois physiques » ou qui expriment des contraintes sur les propriétés des composants en fonction des relations entre composants est appelé « modèle de calcul, (MoC), Model of Computation ».

Généralement, un modèle de calcul doit expliquer le comportement d'un système pendant son traitement qui peut s'effectuer de manière cyclique, réactive, concurrente ou séquentielle. Il doit également fournir le protocole de communication utilisé pour les interactions avec son environnement. Ce protocole de communication peut être le rendez-vous, le passage de messages, l'échange d'événements, etc. De même un modèle de calcul doit spécifier le format de données à utiliser qui peuvent être des événements, des requêtes, des flux, etc.

Dans les paragraphes suivants, nous présentons quelques modèles de calcul les plus utilisés à savoir : temps continu (CT : Continous Time), temps discret (DT : Discret Time), événements discrets (DE : Discrete Event), flot de données (DF : Data Flow), flot de données synchrones (SDF : Synchronous Data Flow).

III.1.1. Temps continu (CT)

Le modèle de calcul de temps continu [I-14] est utilisé dans des systèmes qui utilisent une dynamique continue, c'est-à-dire, dans les systèmes disposant de composants interagissant par l'intermédiaire des signaux à temps continu. Ces composants impliquent des relations algébriques ou différentielles entre leurs entrées et leurs sorties.

Ce type de systèmes est utilisé par exemple pour la spécification des circuits analogiques. La simulation de ce type de système implique la résolution numérique des équations différentielles.

Les techniques usuelles d'intégration déterminent un pas d'intégration (incrément du temps entre deux valeurs calculées) permettant d'obtenir une précision donnée sur la valeur des

fonctions du temps. Lorsqu'on souhaite de plus avoir une certaine précision sur la date à laquelle une fonction passe par une valeur donnée (par exemple pour générer un événement), il est nécessaire de raffiner l'intégration jusqu'à ce que le pas indique suffisamment précisément la date de l'événement. L'état du modèle est alors obtenu par calcul d'un point fixe.

III.1.2. Temps discret (DT)

Un modèle est à temps discret s'il est conditionné par une horloge qui progresse à une cadence par des sauts réguliers d'une valeur entière à la valeur suivante (un nombre entier représente un multiple d'une certaine unité de temps). La discrétisation du temps revient à parler d'une suite d'itérations qui représente les occurrences d'événements. Ce modèle est particulièrement adapté à la description des fonctions de traitement de données, traitement numérique du signal, où les entrées/sorties sont lues et mise à jour à des instants définis par une loi d'échantillonnage.

III.1.3. Evènements discrets (DE)

Il s'agit en fait d'un système représenté par un modèle à événements discrets [I-15]. L'évolution du système est considérée à des instants de temps particuliers auxquels correspondent des changements significatifs, appelés des évènements.

Dans un modèle à évènements discrets, la synchronisation des acteurs est réalisée au moyen d'évènements apparaissant de manière ponctuelle le long d'un axe temporel généralement continu. Un acteur qui détecte un évènement réagit en changeant d'état ou/et en émettant d'autres évènements. Les événements sont traités dans l'ordre chronologique.

III.1.4. Les systèmes hybrides

Les systèmes hybrides se situent à mi-chemin entre les systèmes continus et les systèmes à événements discrets. Ces systèmes sont des systèmes dynamiques faisant intervenir explicitement et simultanément des phénomènes ou des modèles de type dynamique continue et événementielle.

III.1.5. Flot de données et Flot de données synchrones

Le modèle de calcul Flot de données est utilisé dans des systèmes qui interagissent avec leur environnement en échangeant des flots de données [I-16] [I-17]. L'entrée du système

dispose de files d'attente qui reçoivent une suite de données, et lorsque toutes les entrées sont sollicitées, le système exécute un traitement suivi d'une production éventuelle des données de manière asynchrone. Ces données sont déposées sur la file d'attente de sortie. Deux modèles de flots de données existent :

- **Flot de données synchrone** : le nombre de données consommées et produites est constant d'une invocation à l'autre.
- **Flot de données dynamique** : les blocs sont complètement indépendants et le nombre de données consommées et produites peut varier d'une invocation à l'autre.

III.2. LES DIFFERENTS SIMULATEURS SYSTEME

III.2.1. Matlab/Simulink

Matlab est un environnement d'évaluation de modèles numériques, il inclut une interface avec le langage C permettant une ouverture sur d'autres types de langages. Dans le domaine des systèmes de traitement du signal Matlab est utilisé en conjonction avec l'environnement SIMULINK [I-18], option venant compléter le noyau MATLAB. Simulink fourni une interface graphique pour la modélisation de systèmes dynamiques sous forme de schémas-blocs. Grâce aux nombreux blocs de base fournis, il est possible de créer des modèles rapidement et clairement, sans écrire une seule ligne de code. A partir des modèles Simulink, des fonctions sophistiquées de simulation et d'analyse permettent d'obtenir des résultats rapides et précis : méthodes d'intégration pour systèmes à pas fixe ou variable, simulation interactive avec visualisation des signaux, simulations de Monte-Carlo, définition des points d'équilibre stable, linéarisation, etc.

Le comportement global d'un système Simulink est typiquement le comportement d'un système flot de données. La spécification du système définit les dépendances de données entre les blocs et ainsi un ordre d'exécution de ces derniers de façon à ce que chacun dispose des données lors de son exécution. L'exécution d'un bloc produit des données et permet à un nouveau bloc d'être exécuté. Il faut noter aussi que Simulink utilise les modèles de calcul à temps continu (CT) et à temps discret pour résoudre les équations différentielles.

III.2.2. Scilab/Scicos

Scilab (*contraction de Scientific Laboratory*) [I-19] est un logiciel gratuit, développé à l'INRIA. Scilab utilise un langage propre, semblable dans sa structure aux langages informatiques de haut niveau (Fortran, C, Pascal,…). Il possède en outre un grand nombre d'algorithmes préprogrammés, il est très proche dans ses performances comme dans sa syntaxe, du logiciel Matlab. Comme le logiciel Matlab, Scilab possède une extension semblable à Simulink qui s'appelle Scicos (Scilab connected object simulator). C'est une boite à outils dédiée à la description et à la simulation des systèmes dynamiques hybrides. Plus précisément, Scicos et un environnement de simulation de systèmes, incluant des parties «continues» et «événementielles». Contrairement à d'autres simulateurs de systèmes hybrides, Scicos n'a pas été construit par extension d'un simulateur de systèmes continus ou de systèmes discrets. Scicos est basé sur un formalisme qui prend en compte les deux aspects. Scicos comprend un éditeur graphique de schéma-blocs qui peut être utilisé pour décrire des modèles complexes en connectant des blocs qui représentent des fonctions de base prédéfinies, disponibles dans des «palettes», ou des fonctions utilisateur. Une large classe de systèmes hybrides peuvent être modélisés.

Le comportement global d'un système Scicos est le comportement d'un système conduit par évènement [I-20]. Ceci signifie que l'activation de chaque bloc est explicitement déterminée par un signal d'activation. Cependant pour le mécanisme d'héritage, Scicos ne fournit dans une certaine mesure que le comportement de flots de données.

III.2.3. SPW

Le logiciel SPW (Signal Processing Worksystem) de Cadence [I-21] est un outil de conception assistée par ordinateur (CAO) conçu pour le développement graphique d'architectures dédiées au traitement du signal. C'est un environnement de prototypage virtuel. A ce titre, il permet de valider, à un haut niveau, un système en cours de développement grâce à des modèles logiciels de chacun des composants intervenants dans l'architecture. SPW peut être utilisé pour modéliser puis simuler un système, le déboguer, le réviser, puis le simuler jusqu'à l'obtention de résultats optimaux. Il permet de développer et simuler rapidement des applications complexes.

Le logiciel possède deux modes de simulation différents **SPB_I** (SPB - Interpreted) et **SPB_C** (SPB - Compiled), les deux simulateurs sont semblables et donnent les mêmes résultats

une fois utilisés pour simuler le même schéma fonctionnel. La seule différence réside dans la façon d'exécuter les différentes opérations de simulation.

Le simulateur **SPB_I** fonctionne comme un langage de programmation interprété, c'est à dire qu'il effectue les opérations de simulation les unes après les autres sans compilation préalable.

Le simulateur **SPB_C** se comporte, quant à lui, comme un langage compilé. Il édite, compile et exécute un programme écrit en C. Une conception de système peut être exécutée considérablement plus rapidement avec **SPB_C** qu'avec **SPB_I**, une fois que le programme de simulation est compilé.

SPW (**SPB_I** ou **SPB_C**) est basé sur la simulation à temps discret, il peut fonctionner comme simulateur de flots de données dynamiques (DDF). Ceci signifie que le taux relatif auquel un bloc traite des échantillons de données peut changer pendant une simulation, selon des conditions indiquées dans le schéma fonctionnel.

IV. Les facteurs de mérite de simulation système

La simulation système est un domaine très vaste. Nous nous intéressons dans ce paragraphe uniquement à la simulation du bloc analogique RF et micro-ondes et notamment l'amplificateur de puissances, qui est en règles générale le seul bloc pris en considération dans la simulation système.

Dans une chaîne de communication, l'amplificateur de puissance, module non linéaire est parmi les dispositifs les plus critiques car il affecte notablement les performances globales d'un système en terme d'intégrité du signal et de consommation.

Nous présentons dans ce paragraphe, les concepts généraux qui constituent les figures de mérite de ces amplificateurs (C/I, NPR, ACPR …).

IV.1.1. Intermodulation d'ordre 3 (C/I_3)

L'intermodulation se produit lorsque des signaux de fréquences différentes sont appliqués à un amplificateur ou un système non linéaire. A la sortie de celui-ci se trouvent des signaux qui ne sont pas des harmoniques des fréquences d'origine mais une combinaison de ces fréquences. Dans un système de communication, ceci signifie que les signaux dans un canal peuvent causer l'interférence avec les canaux adjacents, ces effets peuvent être néfastes pour le système car ces interférences peuvent produire une intermodulation qui se superpose au signal utile ou dans la bande adjacente.

Considérons le cas où le signal d'entrée est composé de deux signaux sinusoïdaux de même amplitude et de fréquences différentes, le signal d'entrée est décrit par l'expression suivante :

$$x(t) = A.cos(2\pi.f_1.t) + A.cos(2\pi.f_2.t) \qquad \text{(I-8)}$$

Un tel signal produit en sortie des produits d'intermodulation du type $m.f1 + n.f2$, dans la bande adjacente et comme ils ne peuvent pas être filtrés, ils viennent perturber irrémédiablement le signal.

La figure I-6 représente le spectre de sortie d'une nonlinéarité excitée par un signal constitué de deux porteuses pures. Nous y avons fait figurer uniquement les produits d'intermodulation jusqu'à l'ordre 3.

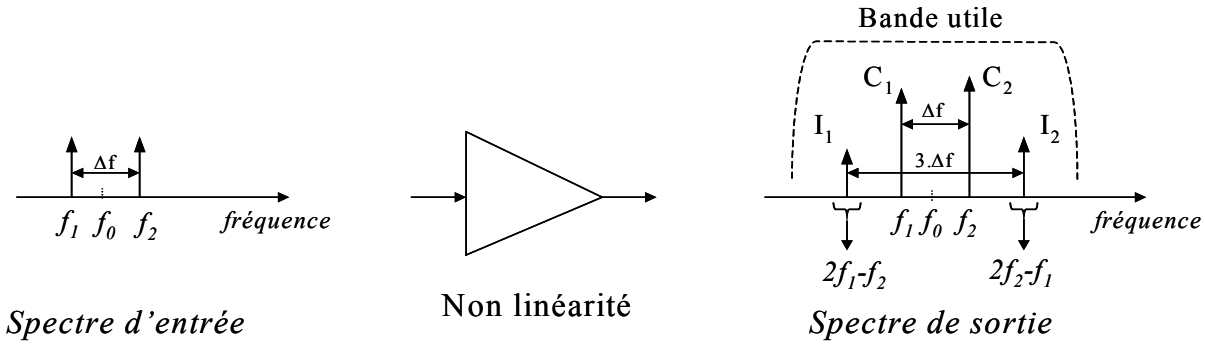


Figure I-6 : Spectre en entrée et en sortie d'une non-linéarité

Le rapport C/I_3 est défini par le rapport entre la puissance des raies porteuses et la puissance des raies d'intermodulation d'ordre 3 comme ci-dessous :

$$C/I_3\left(dB\right) = 10.log\left(\frac{\left|C_1\right|+\left|C_2\right|}{\left|I_1\right|+\left|I_2\right|}\right) \tag{I-9}$$

IV.1.2. L'NPR : Noise Power Ratio

Le signal de test biporteuse n'est pas représentatif pour des applications de télécommunications fonctionnant en multiporteuse. Il est alors naturel d'appliquer à un amplificateur un signal qui comporte un grand nombre de porteuses, normalement indépendantes et régulièrement espacées dans la bande considérée.

Le principe de ce test consiste, à introduire un « trou » au centre de la bande du spectre d'entrée, de manière à évaluer le bruit d'intermodulation présent en sortie généré par la non-linéarité de l'amplificateur. En sortie de l'amplificateur, nous pouvons détecter la partie des produits d'intermodulation générée par la non linéarité et retombant dans le trou préalablement établi comme illustré sur la figure I-7.

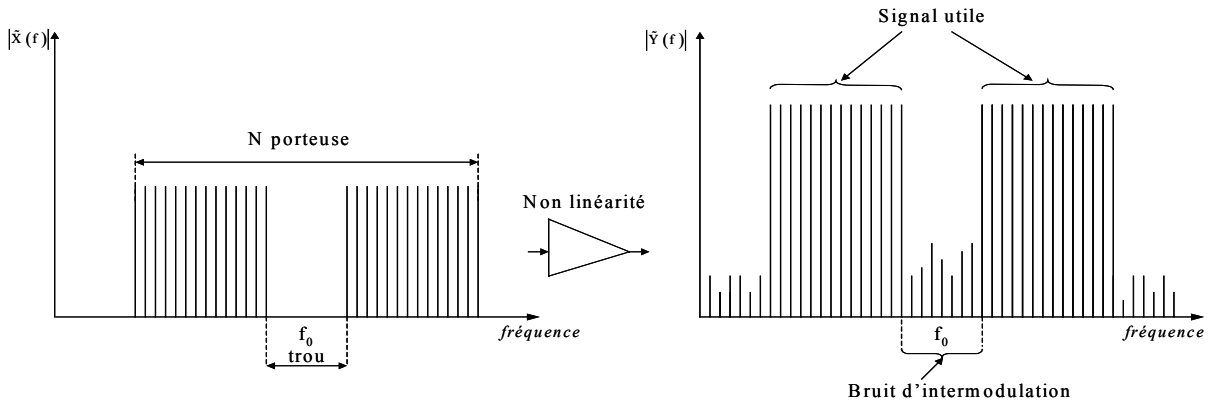


Figure I-7 : Spectre typique du NPR avant et après amplification

Le rapport entre la puissance moyenne du signal utile et celle du bruit d'intermodulation qui remplit le trou dans le spectre de sortie s'appelle le NPR [I-22].

$$\text{NPR } (dB) = 10.\log\left(\frac{\int_{porteuses} P(f).df}{\int_{trou} P(f).df}.\frac{BW_{trou}}{BW_{porteuses}}\right) \tag{I-10}$$

IV.1.3. L'ACPR : Adjacent Channel Power Ratio

Pour plusieurs normes de transmission actuelles (Is-95 CDMA, CDMA2000, WCDMA), l'ACPR (Adjacent Channel Power Ratio) [I-23] est un facteur de mérite important pour quantifier la dégradation du signal de systèmes de communication. Ce rapport est défini comme étant le rapport entre la puissance du signal dans le canal utile et la puissance du signal généré par les distorsions dans un canal adjacent.

$$\text{ACPR } (dB) = 10.\log\left(\frac{2.\int_{BW_0} P(f).df}{\int_{BW_1} P(f).df + \int_{BW_2} P(f).df}\right) \tag{I-11}$$

Tout comme pour l'intermodulation d'ordre 3, nous pouvons définir un ACPR droit, rapport du canal adjacent droit sur la puissance dans le canal principal, ainsi qu'un ACPR gauche, rapport du canal adjacent gauche sur la puissance dans le canal principal.

La figure I-8 présente les perturbations introduites sur un spectre par un système non linéaire.

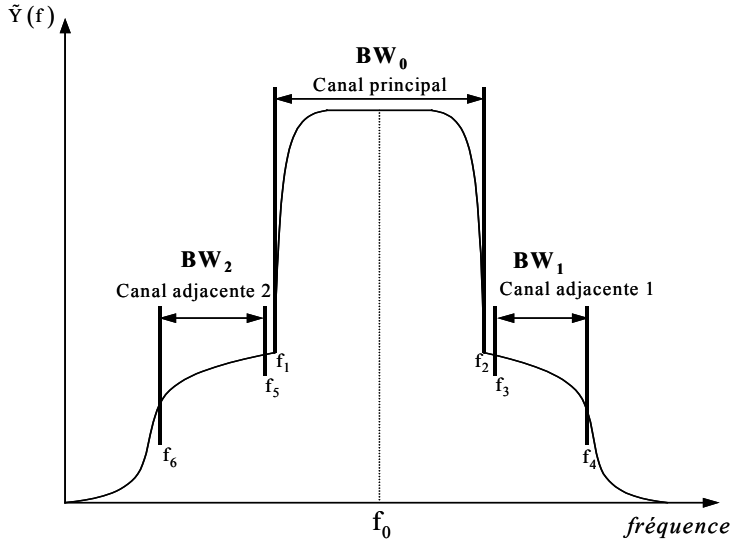


Figure I-8 : Spectre en sortie d'une non-linéarité

IV.1.4. L'EVM : Error Vector Measurement

L'EVM évalue les effets de distorsions d'amplitude et de phase dans les systèmes de communication sur le diagramme de constellation, elle représente la différence entre le signal mesuré (distordu, filtré, bruité, mal synchronisé,…) et une référence (un signal idéalement modulé) dans un diagramme I/Q [I-24] [I-25]. La figure I-9 définit l'EVM et les termes qui lui sont habituellement associés.

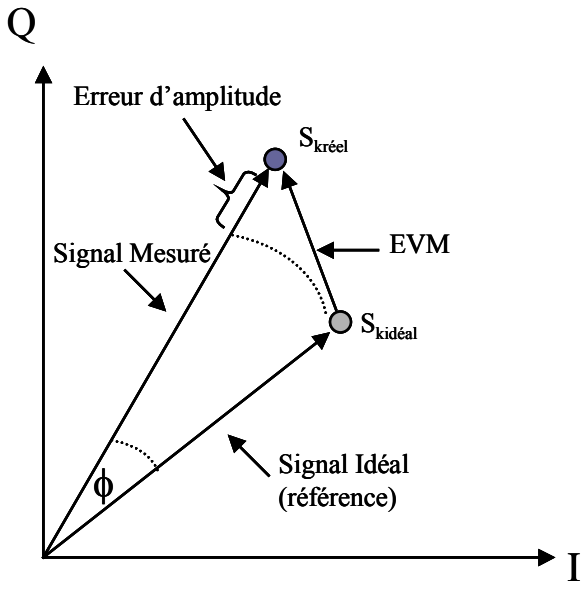


Figure I-9 : Principe de calcul de l'EVM pour un symbole d'une modulation numérique

Le calcul de l'EVM permet d'avoir une idée assez précise de la qualité d'un signal numérique modulé. On la calcule en fin de simulation de l'Enveloppe à partir de deux diagrammes de constellation (celui qui correspond au signal réel et celui qui correspond au signal idéal). En pratique, on dispose des composantes I et Q du signal mesuré et du signal de référence à chaque temps de rendez-vous (lors d'un changement de symbole).

On calcule alors l'EVM comme suit :

$$EVM = \sqrt{\frac{\frac{1}{N}\cdot\sum_{k=1}^{N}\left(\left|S_{kideal} - S_{kréel}\right|^2\right)}{\frac{1}{N}\cdot\sum_{k=1}^{N} S_{kidéal}^2}}.100\% \qquad \text{(I-12)}$$

$S_{kidéal}$: Symbole idéal numéro k de la constellation.

$S_{kréel}$: Symbole réel associé à $S_{kidéal}$.

N : Nombre de symboles nécessaires pour la modulation.

IV.1.5. Taux d'Erreur Binaire (TEB)

Le TEB mesure le taux d'erreur lors du décodage de l'information [I-26]. C'est le rapport du nombre de symboles erronés sur le nombre de symboles de la séquence d'entrée. C'est une statistique utile pour évaluer la qualité d'un système de communication. Toutefois, elle nécessite un nombre important de symboles (de plusieurs milliers à plusieurs millions), d'où un temps de simulation très long.

Pour une modulation MPSK, l'information est contenue exclusivement dans la phase du signal. On considère que celle-ci est une variable aléatoire continue à valeurs réelles, notée θ, qui suit une loi normale de moyenne nulle et d'écart type σ égal à l'erreur de phase. Un point dans le diagramme de constellation réel est interprété comme le symbole représenté par le point du diagramme de constellation de référence le plus proche en phase. C'est pourquoi on définit l'angle de décision t_d comme étant la moitié de l'angle existant entre deux positions idéales adjacentes dans le diagramme de constellation de référence (Figure I.10). Si l'on note M le nombre de positions idéales (aussi dénommé nombre d'états), on calcule alors le TEB comme suit :

$$TEB_M = P\left(|\theta| > t_d\right) = \begin{cases} 1 - erf\left(\dfrac{\pi}{M\sigma\sqrt{2}}\right) & si \quad \sigma > 0 \\ 0 & si \quad \sigma = 0 \end{cases} \tag{I-13}$$

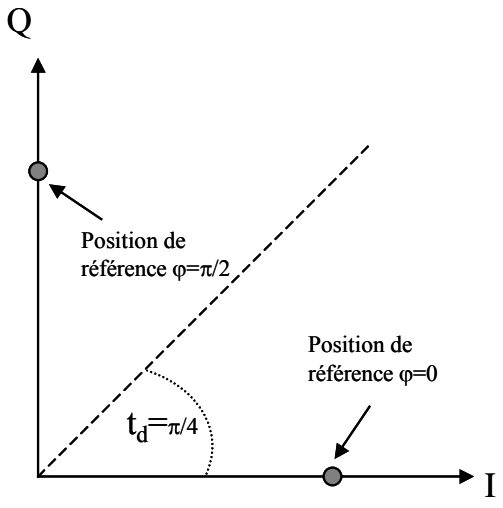


Figure I-10 : Définition de l'angle de décision pour une modulation QPSK.

V. La modélisation comportementale des amplificateurs de puissance pour la simulation système

De par la gestion délicate du compromis entre sa consommation électrique et les distorsions induites par sa non-linéarité, l'amplificateur de puissance représente un élément critique dans le calcul de bilan de liaison des systèmes de télécommunications. Cependant les modèles utilisés couramment dans les simulateurs système sont généralement des modèles relativement peu précis, qui ne rendent pas bien compte des imperfections importantes comme les effets de mémoire non linéaires et les désadaptions d'impédance. Récemment un effort de recherche important a été réalisé sur la prise en compte de ces imperfections dans les modèles comportementaux pour la simulation système. Des nombreux résultats très intéressants ont été publiés mais malheureusement ces modèles ne sont pas encore disponibles dans les environnements de simulation généralistes.

Dans cette partie nous proposons une brève revue de ces techniques de modélisation qui ont constitué le point de départ de notre travail. Nous nous attachons plus particulièrement dans cette partie au domaine de la modélisation des modules hyperfréquences non-linéaires. Nous ne rentrerons pas dans le détail des différents modèles, notre préoccupation est ici de bien identifier les procédures de caractérisation des divers modèles ainsi que leurs équations constitutives, de façon à pouvoir étudier une implémentation efficace dans les simulateurs système.

V.1. FORMALISME D'ENVELOPPE COMPLEXE

Les signaux manipulés dans les systèmes de télécommunication sont généralement des signaux de type passe-bande. En général les amplificateurs possèdent des filtres en entrée et sortie, permettant de ne garder que la partie du spectre utile donc du spectre autour de la fréquence fondamentale de la porteuse, comme indiqué sur la figure I-11.

Dans ces conditions, les signaux présentés aux accès du dispositif peuvent s'écrire sous la forme suivante :

$$x(t) = Re\left(\tilde{X}(t).e^{j\omega_0 t}\right) \tag{I-14}$$

où $x(t)$ est un signal réel tel que $x(t) \in \mathbb{R}$.

$\tilde{X}(t)$ est l'enveloppe complexe du signal réel telle que $\tilde{X}(t) \in \mathbb{C}$.

Pour faciliter les calculs mathématiques, nous pouvons introduire la notion de fréquence négative, ce qui conduit à exprimer l'équation I-14 sous la forme équivalente suivante :

$$x(t) = \frac{1}{2}\tilde{X}(t).e^{j\omega_0 t} + \frac{1}{2}\tilde{X}^*(t).e^{-j\omega_0 t} \qquad \text{(I-15)}$$

où $\tilde{X}^*(t)$ est l'enveloppe complexe conjuguée du signal $\tilde{X}(t)$.

La formulation en enveloppe complexe offre sur le signal réel des avantages non négligeables sur les temps de calcul parce que la fréquence d'échantillonnage est réduite à $f_e \geq BW$ au lieu d'être $f_e \geq 2\left(f_0 + \frac{BW}{2}\right)$ [I-27].

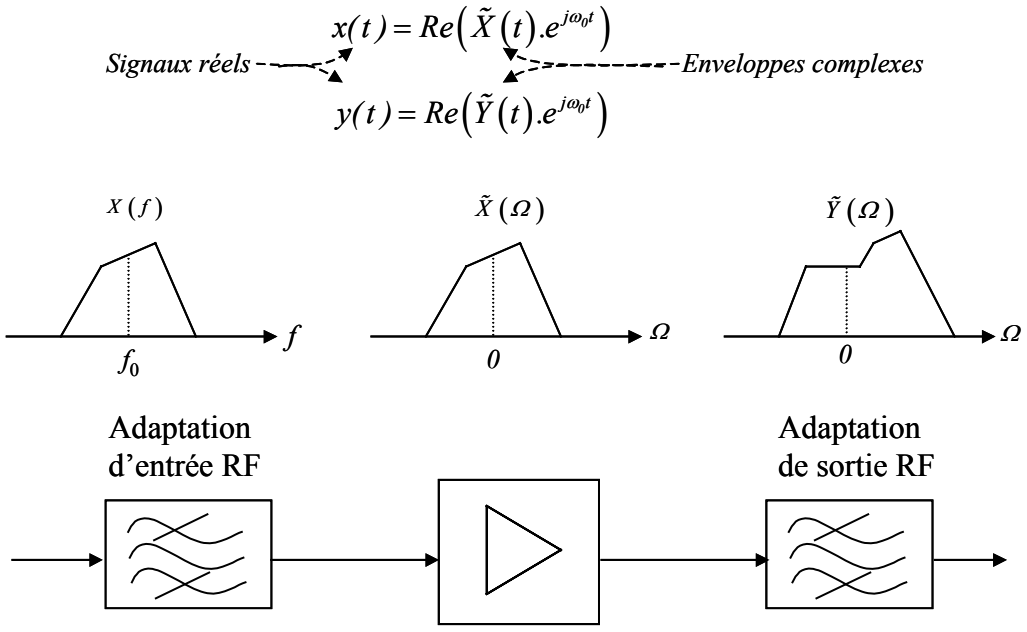


Figure I-11 : **Description en bande de base de modèles non linéaires**

V.2. DESCRIPTION BANDE ETROITE DES MODELES BOITE NOIRE

Le modèle boite noire ou comportemental peut se représenter comme sur la figure I-12. Dans le cadre d'une simulation d'enveloppe complexe, le rôle du modèle boite noire, se borne à identifier la relation mathématique entre les enveloppes complexes du signal d'entrée et du signal de sortie en faisant abstraction des connaissances physique sur la dynamique interne du système. Cette démarche peut avoir différentes raisons : les phénomènes mis en jeu s'avèrent trop complexes à décrire précisément et engendrent un coût d'extraction du modèle trop important, ou encore la description à l'aide des lois physiques est incompatible avec l'utilisation du modèle pour des raisons de coût de calcul.

Les variables $\tilde{X}(t)$ et $\tilde{Y}(t)$ représentent alors les enveloppes complexes d'entrée et de sortie, ω_0 est la fréquence porteuse. L'entrée et la sortie peuvent être définis de multiples façons tensions, courants ou ondes de puissance, comme nous allons voir ci-dessous.

Figure I-12 : Représentation d'un système par son entrée et sa sortie

V.2.1. Application à la modélisation des amplificateurs

Soit le schéma de principe d'un quadripôle ci-dessous, ce quadripôle peut être décrit par des ondes de puissance incidentes et réfléchies aux accès de l'amplificateur ou par les ondes de courant (*i* et *v*).

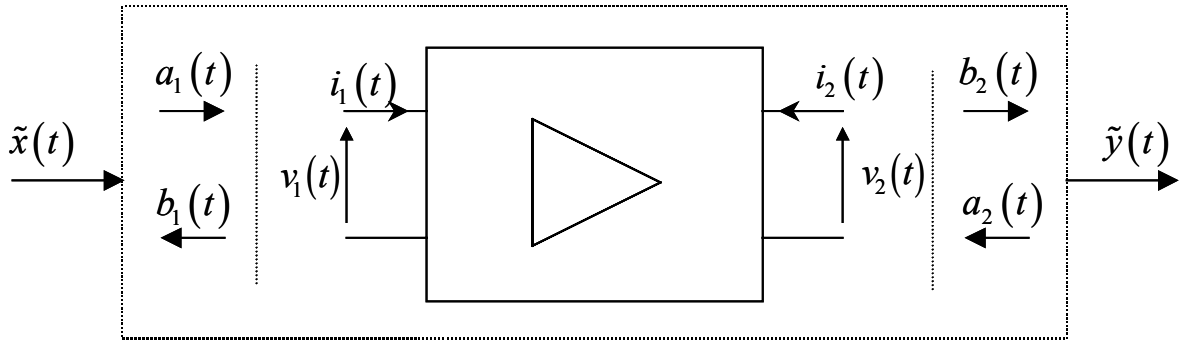


Figure I-13 : Schéma de principe d'un quadripôle non linéaire

L'objectif de la modélisation est de trouver les relations liant le vecteur d'entrée $\tilde{x}(t)$ et le vecteur de sortie $\tilde{y}(t)$ en enveloppe complexe. Si nous voulons modéliser un modèle bilatéral, le vecteur $\tilde{x}(t)$ est défini par $a_1(t)$ et $a_2(t)$ et le vecteur de sortie par $b_1(t)$ et $b_2(t)$.

Les équations reliant les ondes de puissance aux courants et tensions sont données ci-dessous :

$$a_1(t) = \frac{v_1(t) + z_0.i_1(t)}{\sqrt{2.\left[z_0 + z_0^*\right]}} \qquad b_1(t) = \frac{v_1(t) - z_0.i_1(t)}{\sqrt{2.\left[z_0 + z_0^*\right]}}$$

$$a_2(t) = \frac{v_2(t) + z_0.i_2(t)}{\sqrt{2.\left[z_0 + z_0^*\right]}} \qquad b_2(t) = \frac{v_2(t) - z_0.i_2(t)}{\sqrt{2.\left[z_0 + z_0^*\right]}}$$

Avec z_0 l'impédance de référence complexe souvent prise égale à 50 Ω.

La modélisation consiste à déterminer les relations $b_1(t) = f_{NL}\left(a_1(t), a_2(t)\right)$ et $b_2(t) = f_{NL}\left(a_1(t), a_2(t)\right)$, qui prennent en compte la dépendance par rapport aux impédances de fermeture.

Pour simplicité, nous allons nous limiter dans ce qui suit au cas de modules adaptés sur l'impédance de référence z_0 = 50 Ohms. Dans ce cas les ondes de puissance $b_1(t)$ et $a_2(t)$ sont nulles, il suffit alors de trouver la relation unilatérale reliant directement la sortie à l'entrée.

Le schéma équivalent de ce modèle dit unilatéral est représenté sur la figure suivante :

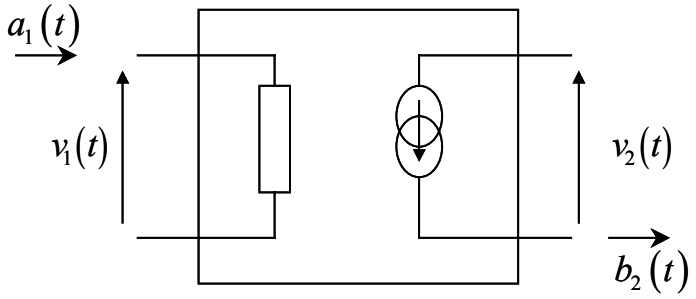


Figure I-14 : Schéma du modèle unilatéral

Dans telles conditions les courants entrant dans le quadripôle sont directement proportionnels aux tensions ($I_1=V_1/50$, $I_2=-V_2/50$). De même, la relation entre les ondes de puissances et les tensions (dans le domaine temporel ou fréquentiel) se simplifie et nous obtenons directement :

$$a_1(t) = \frac{\sqrt{2}.v_1(t)}{\sqrt{z_0}} \qquad b_2(t) = \frac{\sqrt{2}.v_2(t)}{\sqrt{z_0}}$$

V.3. LES MODELES A TOPOLOGIE LOCALISEE

Les premiers modèles ayant été mis en œuvre sont les modèles de topologie localisée. Ils ont été développés à l'origine pour les amplificateurs à Tube à Onde Progressive (TOP) et ils sont basés sur des approches empiriques ; on considère que la structure de l'amplificateur est divisée en deux parties : la première traitant la non linéarité statique et la deuxième les effets mémoire.

Nous allons, tout d'abord présenter le modèle sans mémoire, puis les modèles à topologie localisée usuelles qui permettent la prise en compte de la mémoire.

V.3.1. Modèles sans mémoire

Un modèle non linéaire sans mémoire est un modèle dont les constants de temps sont très inférieure à l'inverse de la fréquence maximale présentée dans le signal d'entrée. Dans un tel modèle, la distorsion d'amplitude et de phase dépend uniquement du niveau du signal présent au même instant. Un tel modèle souvent qualifié de « modèle de gain complexe » peut être modélisé précisément à partir de deux caractéristiques dites courbes AM/AM et AM/PM, comme illustré sur la figure suivante.

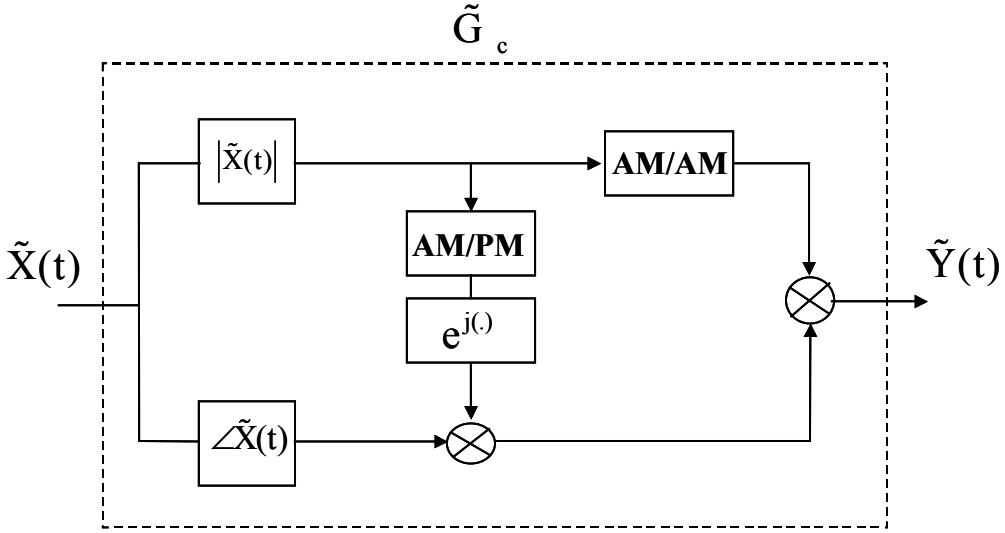


Figure I-15 : Schéma de principe du modèle sans mémoire

La sortie de l'amplificateur a la forme suivante :

$$\tilde{Y}(t) = \tilde{G}_c\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t) \tag{I-16}$$

où $\tilde{G}_c\left(\left|\tilde{X}(t)\right|\right) = \left|\tilde{G}_c\left(\left|\tilde{X}(t)\right|\right)\right| e^{j\varphi\left(\left|\tilde{X}(t)\right|\right)}$.

L'amplitude et la phase du gain complexe $\tilde{G}_c(\bullet)$ représentent respectivement les courbes AM/AM et AM/PM. Ces courbes peuvent être approximées par des polynômes, les séries de fonction sinusoïdale, ou les séries de Bessel [I-28].

Ce modèle peut être bien adapté pour les signaux bande étroite, en revanche lorsque nous appliquons des signaux de bande passante importante, l'hypothèse « sans mémoire » ne peut plus être valable.

V.3.2. Modèles à mémoire

Le modèle sans mémoire que nous avons présenté dans le paragraphe précédent ignore la dépendance fréquentielle des caractéristiques de l'amplificateur. Dans ce paragraphe, nous allons présenter les modèles qui tentent d'intégrer la notion de fréquence. Il est possible de faire une distinction entre deux types de mémoire non-linéaire, la mémoire haute fréquence et la mémoire basse fréquence.

V.3.2.1. Modèles à mémoire haute fréquence

Les effets hautes fréquence dans les amplificateurs non linéaire interviennent, lorsque le dispositif est soumis à un signal d'excitation purement monoporteuse. Ces effets HF se définissent comme la réponse fréquentielle du dispositif sur toute la bande passante. Ils sont mis en évidence par la variation des caractéristiques AM/AM et AM/PM en fonction de la fréquence. Ce type de mémoire est caractérisé par des constantes de temps relativement courtes, de l'ordre de grandeur de la rapidité du signal hyperfréquence. L'origine de ce type de mémoire se situe dans les circuits d'adaptation des transistors et aussi dans la dynamique interne du transistor (temps de transit).

Les modèles dits à « mémoire haute fréquence » essaient de reproduire la variation des courbes AM/AM et AM/PM dans la bande de fonctionnement de l'amplificateur. Ces variations traduisent en fait la mémoire liée à la porteuse.

V.3.2.1.1. Modèle d'Hammerstein

Afin de rendre le modèle dépendant de la fréquence, il faut faire intervenir la fréquence dans la relation (I-16). La solution adoptée dans le modèle d'Hammerstein [I-29] consiste à placer après le modèle non linéaire statique présenté dans la section précédente, un filtre linéaire en sortie comme illustré sur la figure I-16. On remarque cependant que ce type de modèle ne permet de reproduire que des courbes AM/AM, AM/PM ayant une courbure uniforme par rapport à la puissance d'entrée.

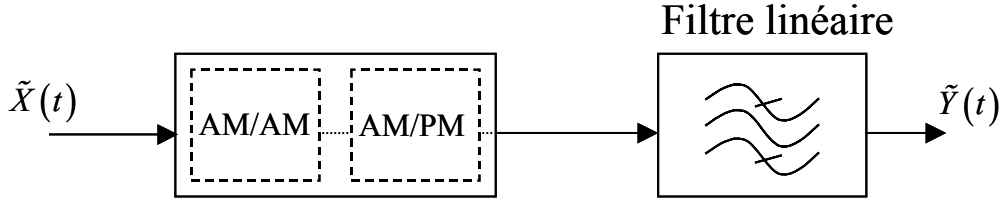


Figure I-16 : Modèle d'Hammerstein

V.3.2.1.2. Modèle de Wiener

La structure du modèle de Wiener est composée d'un filtre linéaire en entrée suivi d'une non linéarité statique extraite à la fréquence centrale. Ce modèle permet de prendre en compte les variations de courbure différentes en fonction de la fréquence d'un niveau de puissance à un autre, mais la généralité du modèle reste faible.

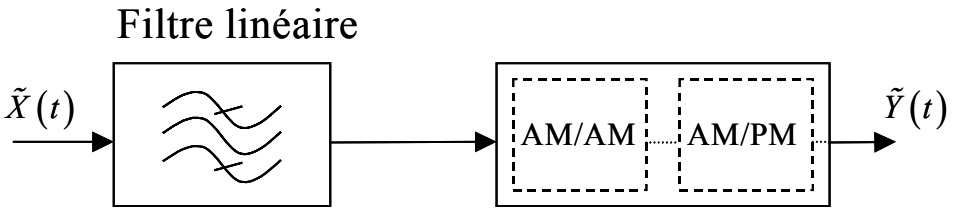


Figure I-17 : Modèle de Wiener

V.3.2.2. Modèles à mémoire basse fréquence

Les effets basses fréquence interviennent essentiellement lorsque le dispositif est soumis à un signal d'excitation complexe. Ces effets BF sont néanmoins présents en fonctionnement purement monoporteuse, mais n'influencent pas directement la porteuse en régime établi. Ce type de phénomène non-linéaire est caractérisé par des constantes de temps très importantes par rapport à la vitesse du signal hyperfréquence. L'origine de ce type de mémoire se trouve dans les constantes de temps introduites par les circuits de polarisation et les réseaux d'adaptation, de même que dans l'interaction électrothermique des semiconducteurs.

Pour décrire ce type de mémoire, Bösch [I-30] propose un modèle qui tient compte des effets de mémoire basse fréquence, ce modèle est constitué de deux voies, la première modélisant le comportement haute fréquence de l'amplificateur à différents points de polarisation, et une seconde voie modélisant la réaction basse fréquence due au circuit de polarisation comme illustré dans la figure I-18.

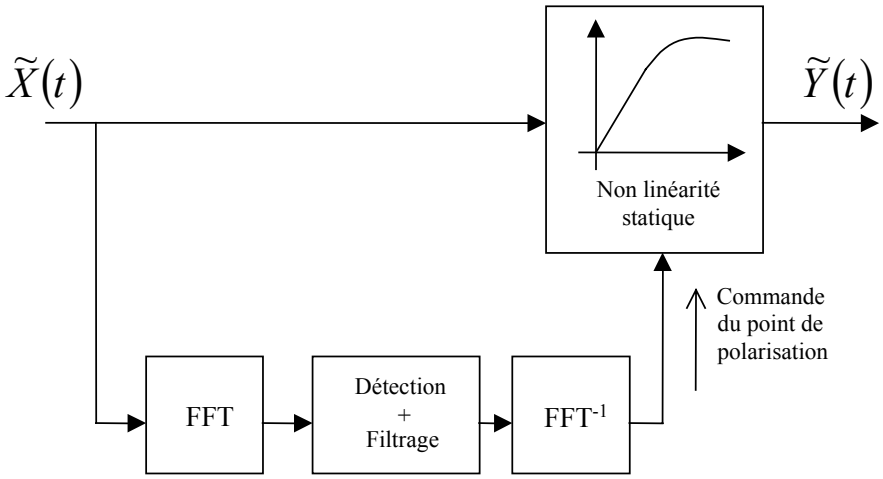


Figure I-18 : Modèle de Bösch

Les trois modèles à mémoire décrits ci-dessus donnent de résultats plus ou moins satisfaisants en fonction des applications et des technologies d'amplificateurs, leur limitation principale vient de ce qu'ils supposent que le système à mémoire puisse se décomposer en une combinaison très restreinte d'éléments statiques non linéaires et purement linéaires.

V.4. LES MODELES A TOPOLOGIE DISTRIBUEE

Les modèles à topologie localisée présentés précédemment sont basés sur des structures prédéfinies. Le modèle d'amplificateur est devisé en deux parties, la partie linéaire à mémoire modélisée par des filtres linéaires et la partie non linéaire sans mémoire. Plusieurs travaux ont été menés ces dernières années pour développer des modèles qui ne reposent pas sur une topologie empirique prédéfinie, mais qui s'appuient sur des bases mathématiques, dans l'objectif d'obtenir des modèles plus généraux, capables de décrire les effets de mémoire à court et long termes indépendamment des technologies d'amplificateur.

On peut citer parmi ces travaux, le modèle de Volterra à un et deux noyaux basé sur le formalisme des séries de Volterra à noyaux dynamiques [I-31], le modèle à réponse impulsionnelle non linéaire [I-32] et le modèle à séries de Volterra « modulées » [I-33], nous allons résumer dans la suite de ce chapitre le principe de ces trois modèles.

V.4.1. Formalisme mathématique des modèles basés sur les séries de Volterra

V.4.1.1. Rappel sur les séries de Volterra classique

La théorie des systèmes linéaires enseigne que la réponse à un signal *x(t)* peut être calculée à l'aide de l'intégrale de convolution suivante :

$$y(t) = \int_{-\infty}^{+\infty} h(t-\tau).x(\tau).d\tau = h(t) * x(t) \qquad \text{(I-17)}$$

où *h(t)* est la réponse du système à une impulsion $\delta(t)$ d'une durée infinitésimale.

Cette opération de convolution, nous permet de calculer la réponse d'un système linéaire à mémoire par simple convolution entre le signal appliqué en entrée et sa réponse impulsionnelle. L'extension de ce principe au cas des systèmes non linéaires conduit à la série mise au point par Vito Volterra [I-34] [I-35], connue sous le nom de série de Volterra et s'exprimant comme ci-dessous.

$$y(t)=\sum_{n=1}^{+\infty} y_n(t) \quad où \quad y_n(t)=\int_0^{+\infty}\int_0^{+\infty}\cdots\int_0^{+\infty} h_n(\tau_1,\tau_2,\cdots\tau_n).x(t-\tau_1).x(t-\tau_2).\cdots.x(t-\tau_n).d\tau_1.d\tau_2.\cdots d\tau_n \qquad \text{(I-18)}$$

Dans cette série dite série de Volterra classique, $h_n(\tau_1,\tau_2,\cdots\tau_n)$ représente le noyau de Volterra d'ordre *n* ou réponse impulsionnelle non linéaire. La transformée de Fourier de $h_n(\tau_1,\tau_2,\cdots\tau_n)$ est appelée la fonction de transfert non linéaire. Chaque noyau de la série de Volterra $h_n(\tau_1,\tau_2,\cdots\tau_n)$ représente un invariant du système indépendant du signal d'excitation [I-36].

La figure I-19 représente la structure de la série de Volterra classique, où chaque cellule symbolise un filtre de dimension *n* = 1,2, ….

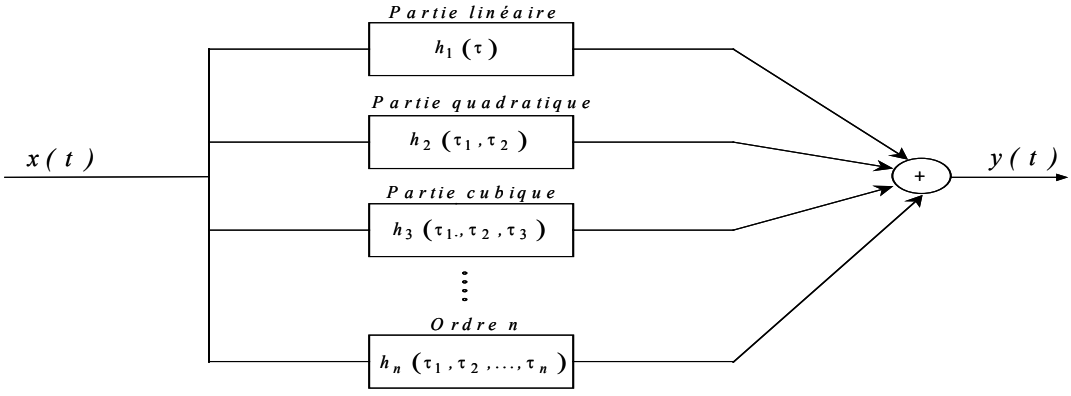


Figure I-19 : Représentation schématique d'une série de Volterra classique

Le champ d'application de ce formalisme rigoureux est malheureusement limité aux dispositifs présentant de faibles non linéarités ou pour de condition d'excitation petit signal, car l'identification des noyaux de Volterra d'ordre supérieur à deux est extrêmement difficile dans la pratique.

V.4.1.2. Séries de Volterra dynamique

Afin de s'affranchir des difficultés évoquées précédemment, Asdente [I-37] puis Filicori, [I-38] [I-39] ont proposé une série de Volterra modifiée permettant de dissocier les phénomènes de saturation pure et les phénomènes de mémoire non linéaire. De sorte il est possible de caractériser des fonctions fortement non linéaires avec un nombre de noyaux réduit, dès lors que les effets de mémoire sont faiblement non linéaires. La série modifiée, dite série de Volterra à noyaux dynamiques se présente comme ci-dessous.

$$y(t) = f\left(x_0,\cdots,x_0\right) + \sum_{n=1}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} h_n\left(x(t),\tau_1,\cdots,\tau_n\right) \prod_{i=1}^{n} \left(x\left(t-\tau_i\right) - x\left(t\right)\right) d\tau_i \qquad \text{(I-19)}$$

$f\left(x_0,\cdots,x_0\right)$ représente la caractéristique statique du système et $h_n\left(x(t),\tau_1,\cdots,\tau_n\right)$ le noyau de Volterra dynamique d'ordre *n*. Contrairement à la série de Volterra classique, les noyaux ne sont plus des invariants du système car ils dépendent du niveau de l'excitation *x(t)*.

La série de Volterra à noyaux dynamiques est le résultat d'un déplacement dynamique d'origine de la série de Volterra classique vers le point de fonctionnement statique. La réponse du système est constituée de deux parties, une partie statique et une partie dynamique.

Si le signal d'entrée *x(t)* varie peu pendant la duré de mémoire, le produit $\prod_{i=1}^{n}\left(x(t-\tau_i)-x(t)\right)$ tend rapidement vers 0 pour n > 1 ; ce qui permet de limiter la série au premier terme, conduisant à l'expression suivante :

$$y(t) = y_{stat}\left(x(t)\right) + \int_{-\infty}^{+\infty} h\left(x(t),\tau\right).\left(x(t-\tau)-x(t)\right).d\tau \qquad \text{(I-20)}$$

Si la puissance du signal est faible, l'approximation I-20 est toujours valide quelle que soit la durée de la mémoire.

Si la puissance du signal est importante la validité du modèle sera réduite à la description des systèmes à mémoire courte par rapport à la rapidité du signal.

Lorsque l'on s'intéresse aux signaux d'enveloppe, alors d'après la définition (I-15), la réponse $\tilde{Y}(t)$ est une fonction des deux variables $\tilde{X}(t)$ et $\tilde{X}^*(t)$. En reconsidérant l'équation (I-20) avec ces deux variables $\tilde{X}(t)$ et $\tilde{X}^*(t)$, nous aboutissons au modèle de Volterra dynamique adapté à la simulation système :

$$\begin{aligned}\tilde{Y}(t) = \tilde{Y}_{stat}\left(\tilde{X}(t),\tilde{X}^*(t)\right) &+ \int_{-\infty}^{+\infty} \tilde{h}_1\left(\tilde{X}(t),\tilde{X}^*(t),\tau\right).\left(\tilde{X}(t-\tau)-\tilde{X}(t)\right).d\tau \\ &+ \int_{-\infty}^{+\infty} \tilde{h}_2\left(\tilde{X}(t),\tilde{X}^*(t),\tau\right).\left(\tilde{X}^*(t-\tau)-\tilde{X}^*(t)\right).d\tau\end{aligned} \quad \text{(I-21)}$$

IV.4.1.2.1 **Forme tempo-fréquentielle des séries de Volterra à noyaux dynamique**

Dans l'expression I-21, la notion de la fréquence n'y apparaît pas clairement, nous pouvons exprimer les noyaux dynamiques dans le domaine fréquentiel, en réalisant la transformée de Fourier des noyaux dynamiques suivant τ.

En notant par $\tilde{H}\left(\tilde{X}(t),\tilde{X}^*(t),\Omega\right)$ et $\tilde{X}(\Omega)$ les transformées de Fourier respectives de $h\left(\tilde{X}(t),\tilde{X}^*(t),\tau\right)$ et $\tilde{X}(t)$, on a :

$$\tilde{H}\left(\tilde{X}(t),\tilde{X}^*(t),\Omega\right) = TF\left(\tilde{h}\left(\tilde{X}(t),\tilde{X}^*(t),\tau\right)\right) = \int_{-\infty}^{+\infty} \tilde{h}\left(\tilde{X}(t),\tilde{X}^*(t),\tau\right)e^{-j\Omega\tau}d\tau \quad \text{(I-22)}$$

$$\tilde{X}(\Omega) = TF\left(\tilde{X}(\tau)\right) = \int_{-\infty}^{+\infty} X(\tau)e^{-j\Omega\tau}d\tau \quad \text{(I-23)}$$

Un examen détaillé de l'équation I-21 montre que chacune des parties dynamiques peut s'exprimer sous forme de produits de convolution.

Après quelques lignes de calcul, on aboutit à l'équation fondamentale du modèle de Volterra dynamique développé par N. Le Gallou [I-40].

$$\begin{aligned}\tilde{Y}(t) = \tilde{Y}_{stat}\left(\tilde{X}(t),\tilde{X}^*(t)\right) &+ \frac{1}{2\pi}\int_{-\infty}^{+\infty} \tilde{H}_1\left(\tilde{X}(t),\tilde{X}^*(t),\Omega\right).\tilde{X}(\Omega).e^{j\Omega t}d\Omega \\ &+ \frac{1}{2\pi}\int_{-\infty}^{+\infty} \tilde{H}_2\left(\tilde{X}(t),\tilde{X}^*(t),-\Omega\right).\tilde{X}^*(\Omega).e^{-j\Omega t}d\Omega\end{aligned} \quad \text{(I-24)}$$

où
$$\begin{cases} \tilde{H}_1\left(\tilde{X}(t),\tilde{X}^*(t),\Omega\right)=\tilde{H}_1\left(\tilde{X}(t),\tilde{X}^*(t),\Omega\right)-\tilde{H}_1\left(\tilde{X}(t),\tilde{X}^*(t),0\right) \\ \tilde{H}_2\left(\tilde{X}(t),\tilde{X}^*(t),-\Omega\right)=\tilde{H}_2\left(\tilde{X}(t),\tilde{X}^*(t),-\Omega\right)-\tilde{H}_2\left(\tilde{X}(t),\tilde{X}^*(t),0\right) \end{cases} \quad \text{(I-25)}$$

V.4.2. Le modèle de Volterra deux noyaux

Le modèle obtenu (I-24) fait apparaître deux fonctions de transfert $\tilde{H}_1$ et $\tilde{H}_2$ dépendant à la fois de la fréquence et la valeur instantanée du signal d'entrée. Il est encore possible, du fait de considérations physiques de modifier l'expression (I-24) [I-41].

La forme tempo-fréquentielle finale de la série de Volterra à deux noyaux est décrite dans l'équation suivante :

$$\begin{aligned} \tilde{Y}(t)=\tilde{H}_0\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t)+\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_1\left(\left|\tilde{X}(t)\right|,\Omega\right).\tilde{X}(\Omega).e^{j\Omega t}.d\Omega \\ +\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_2\left(\left|\tilde{X}(t)\right|,-\Omega\right).e^{j.2.\varphi_{\tilde{X}(t)}}.\tilde{X}^*(\Omega).e^{-j\Omega t}.d\Omega \end{aligned} \quad \text{(I-26)}$$

$\tilde{X}(t)$ et $\tilde{Y}(t)$ enveloppe complexe du signal d'entrée et du signal de sortie ;

H_1 et H_2 les noyaux dynamiques d'ordre 1 ;

$\tilde{X}(\Omega)$ transformée de Fourier de l'enveloppe complexe d'entrée ;

BW la bande passante du dispositif à modéliser ; Ω pulsation en bande de base.

La figure I-20 présente la structure du modèle de Volterra à noyaux dynamiques. Il est composé de deux parties. La première partie prend en compte la réponse du dispositif à la fréquence centrale de fonctionnement, la deuxième partie se focalise sur tous les phénomènes dynamiques du dispositif à l'aide de sa forme d'intégrale.

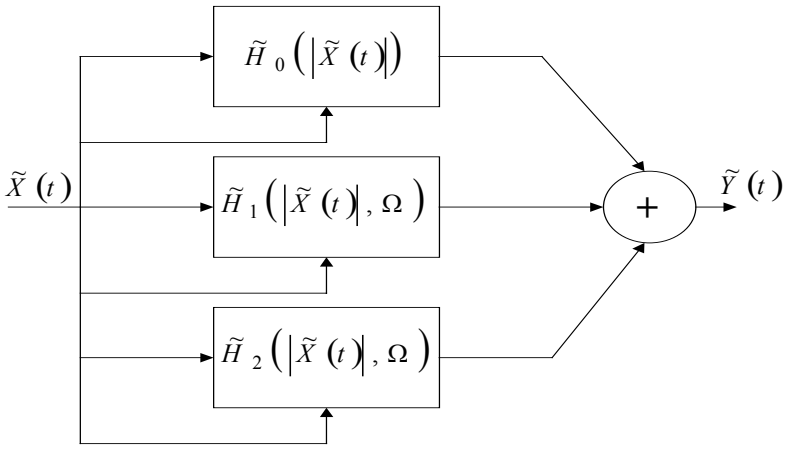


Figure I-20 : Modèle de Volterra à noyaux dynamiques

V.4.2.1. Procédure d'extraction des noyaux de Volterra

L'extraction des noyaux dynamiques de Volterra $\widetilde{H}_1$ et $\widetilde{H}_2$ se fait à l'aide d'un signal composé d'une raie pompe à la fréquence centrale ω_0 (balayant l'ensemble de la caractéristique non linéaire du dispositif) et d'une raie parasite de niveau constant (vu linéairement pas le circuit hyperfréquence) à la fréquence $\omega_0 + \Omega$. A chaque niveau du signal de pompe on balaye la distance Ω entre la fréquence de pompe et la modulation, de façon à couvrir la bande de l'amplificateur à modéliser. Cette procédure d'extraction du modèle de Volterra deux noyaux suit le schéma de principe de la figure ci-dessous.

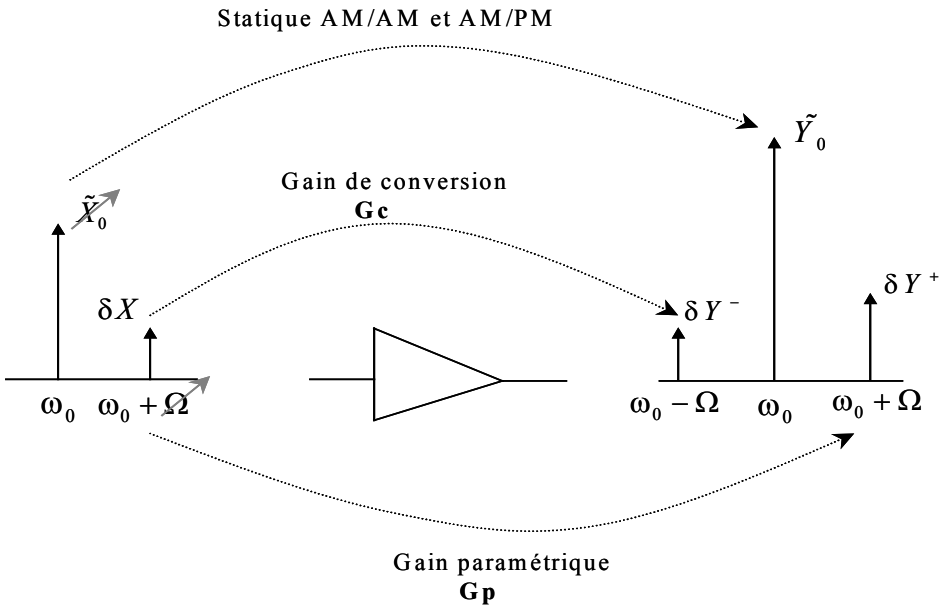


Figure I-21 : Procédure d'extraction du modèle de Volterra deux noyaux

L'application du signal d'entrée $\tilde{X}(t) = \tilde{X}_0 + \delta X.e^{j\Omega t}$ conduit à travers la non linéarité de l'amplificateur à un signal de sortie composé de trois raies de la forme :

$$\tilde{Y}(t) = \tilde{Y}_0 + \delta Y^+.e^{j\Omega t} + \delta Y^-.e^{-j\Omega t} \qquad \text{(I-27)}$$

Pour déterminer l'expression des noyaux dynamiques, nous comparons ce signal à la réponse obtenue avec le modèle. Finalement nous pouvons déduire les expressions analytiques de chacun des noyaux sous la forme [I-42] :

$$\tilde{H}_1(|\tilde{X}_0|,\Omega) = \frac{\delta Y^+}{\delta \tilde{X}} - \frac{1}{2}\left[\frac{\partial Y_0}{\partial|\tilde{X}_0|} + \frac{\tilde{Y}_0}{|\tilde{X}_0|}\right] \qquad \text{(I-28)}$$

$$\tilde{H}_2(|\tilde{X}_0|,\Omega) = \frac{\delta Y^-}{\delta \tilde{X}^*} - \frac{1}{2}\left[\frac{\partial \tilde{Y}_0}{\partial |\tilde{X}_0|} - \frac{\tilde{Y}_0}{|\tilde{X}_0|}\right] \qquad \text{(I-29)}$$

L'identification de ce type de modèle est réalisable de façon simple en simulation par équilibrage harmonique, mais son extraction via des mesures reste délicate. Il faut connaître les relations de phase entre raies spectrales de fréquences différentes. Cela nécessite un principe de caractérisation temporelle calibrée en amplitude et en phase absolue.

A titre d'illustration, un exemple de noyaux dynamiques extraits par simulation d'un amplificateur de puissance HFET fonctionnant en bande C (4 GHz) est présenté sur la figure suivante.

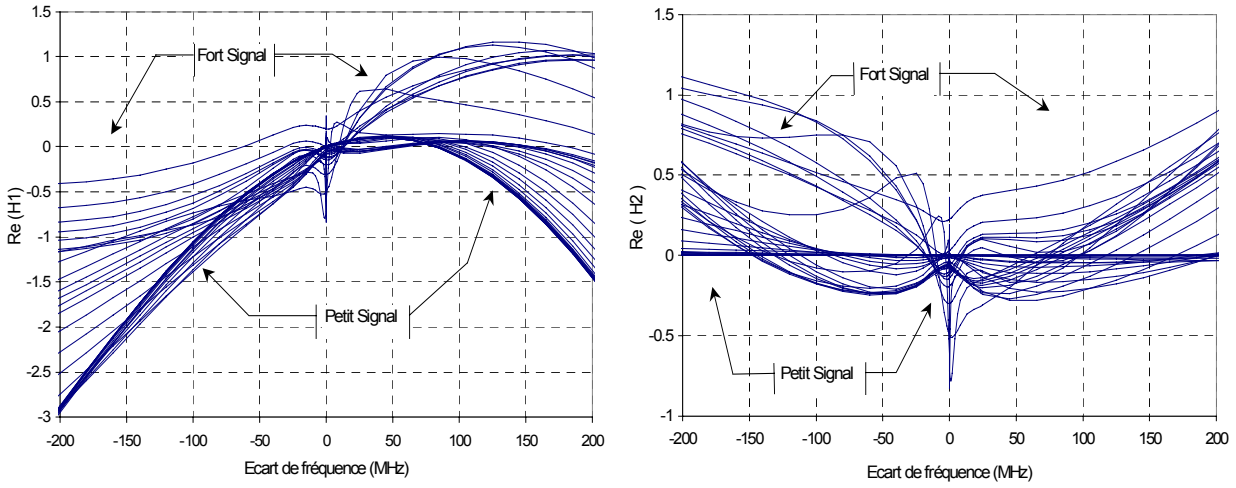


Figure I-22 : Exemple de noyaux dynamiques obtenus par simulation

V.4.3. Le modèle de Volterra un noyau

L'analyse de l'équation de Volterra à deux noyaux montre que dans le cas des applications traitant des signaux à enveloppe quasi-constante comme les systèmes GMSK, le deuxième terme de la série est très peu excité. On peut alors simplifier le modèle qui se réduit à la forme ci-dessous, dite modèle de Volterra dynamique un noyau.

$$\tilde{Y}(t) = \tilde{H}_0\left(|\tilde{X}(t)|\right).\tilde{X}(t) + \frac{1}{2\pi}\int_{-BW/2}^{+BW/2} \tilde{H}_1\left(|\tilde{X}(t)|,\Omega\right).X(\Omega).e^{j\Omega t}.d\Omega \qquad \text{(I-30)}$$

Le principe d'extraction de l'unique noyau dynamique est extrêmement simple, il repose de fait sur une série de mesures des courbes AM/AM et AM/PM où la fréquence est balayée de manière à couvrir la bande utile de l'amplificateur comme le montre la figure I-23.

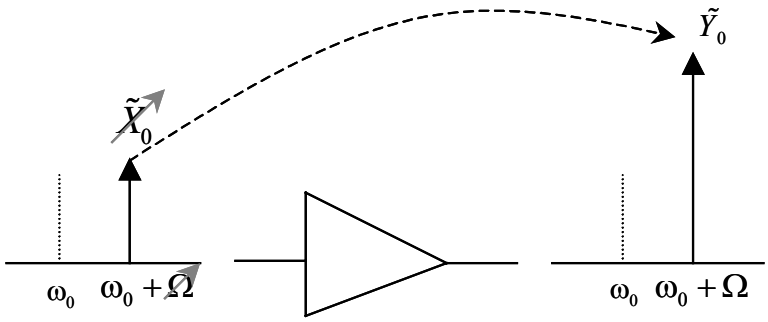


Figure I-23 : Procédure d'extraction du modèle de Volterra un noyau

L'application d'un signal monoporteuse (figure I-23) conduit à travers la non linéarité de l'amplificateur à un signal de sortie composé d'une seule raie $\tilde{Y}_0$ à la même fréquence. Si nous appliquons ce signal dans l'équation fondamentale du modèle, nous aboutissons à l'expression suivante :

$$\tilde{H}_1\left(\left|\tilde{X}_0\right|,\Omega_0\right)=\frac{\tilde{Y}_0}{\tilde{X}_0}-\tilde{H}_0\left(\left|\tilde{X}_0\right|\right) \tag{I-31}$$

Le noyau dynamique $\tilde{H}_1$ constitue la différence, pour une puissance d'entrée donnée, entre la réponse à une fréquence dans la bande utile de l'amplificateur, avec la réponse à la fréquence centrale de référence.

V.4.4. Le modèle à réponse impulsionnelle non linéaire

L'implémentation numérique du modèle de Volterra à noyaux dynamiques ainsi que la complexité d'identification des noyaux obligent à considérer actuellement une limitation de la série au premier ordre. La série ainsi écrite ne permet pas une description des phénomènes de mémoire à long terme fortement non linéaires.

Afin de permettre une prédiction efficace des phénomènes de mémoire longue simulés en présence de signaux à fort taux de modulation d'amplitude, A. Soury [I-43] [I-44] a développé un modèle qui prend en compte les effets de mémoire longue. Le principe de base de ce modèle à réponse impulsionnelle non linéaire est de remplacer les séries de puissance entières (x^n) par des séries de fonctions arbitraires.

Tout comme le modèle de Volterra dynamique, le modèle dit « à réponse impulsionnelle non linéaire » est une forme modifiée de la série de Volterra classique, qui consiste à remplacer les fonctions de base en monômes utilisées dans la décomposition en séries de Volterra classique

par des fonctions arbitraires, qui sont optimisées par rapport à un signal d'extraction. On observe qu'il est alors possible d'obtenir une bonne modélisation des effets de mémoire non linéaire en limitant la série à l'ordre 1. Si l'on considère les notations suivantes :

$$x(t) = \Re e\left\{ \left|\tilde{X}(t)\right| \, . \, e^{\,j.\int_0^t \left(\omega_0 + \frac{d\varphi_{\tilde{X}}(\tau)}{d\tau}\right).d\tau} \right\} \tag{I-32}$$

où $\left|\tilde{X}(t)\right|$ et $\omega_0 + \dfrac{d\varphi_{\tilde{X}}(t)}{dt}$ sont l'amplitude et la pulsation instantanées du signal, le modèle obtenu prend la forme ci-dessous :

$$\tilde{Y}(t) = \int_0^{\infty} \tilde{h}\left(\left|\tilde{X}(t-\tau)\right| \, , \, \left.\frac{d\varphi_{\tilde{X}}(t)}{dt}\right|_{t-\tau} \, , \, \tau \right) . \tilde{X}(t-\tau) \, . \, d\tau \tag{I-33}$$

V.4.4.1. Identification de la réponse impulsionnelle non linéaire

L'identification de la réponse impulsionnelle se fait en excitant le dispositif par une un échelon de Heaviside modulant la porteuse ω_0 (Figure I-24). Cette méthode d'extraction est réalisée par l'intermédiaire de simulation en transitoire d'enveloppe ou de caractérisation expérimentale sur banc de mesure d'enveloppe temporelle. Le signal d'excitation a la forme suivante :

$$x(t) = \Re e\left\{ X_0 . U(t) . e^{j.\omega_0 .t} \right\} \tag{I-34}$$

En remplaçant l'enveloppe complexe du signal (I-34) dans l'expression (I-33), on obtient :

$$\widetilde{Y}(X_0, t) = \int_0^{Tm} \widetilde{h}\{ X_0, \tau \} . X_0 . U(t-\tau) . d\tau \tag{I-35}$$

En utilisant le fait que $U(t-\tau) = 0$ pour $\tau \succ t$ et que $U(t-\tau) = 1$ pour $\tau \leq t$, on peut réécrire la relation (I-35) comme ce qui suit.

$$\widetilde{Y}(X_0, t) = \int_0^{t} \widetilde{h}\{ X_0, \tau \} . X_0 . d\tau \tag{I-36}$$

Nous pouvons déduire l'expression de la réponse impulsionnelle à partir de la relation (I-36), en considérant la dérivée par rapport au temps de l'enveloppe complexe de sortie comme indiqué dans l'équation I-37.

$$\tilde{h}(X_0\ ,\ t) = \frac{1}{X_0}.\frac{\partial \tilde{Y}(X_0\ ,\ t)}{\partial t} \tag{I-37}$$

L'amplitude du signal d'entrée X_0 doit être balayée afin de couvrir toute la plage de fonctionnement du dispositif.

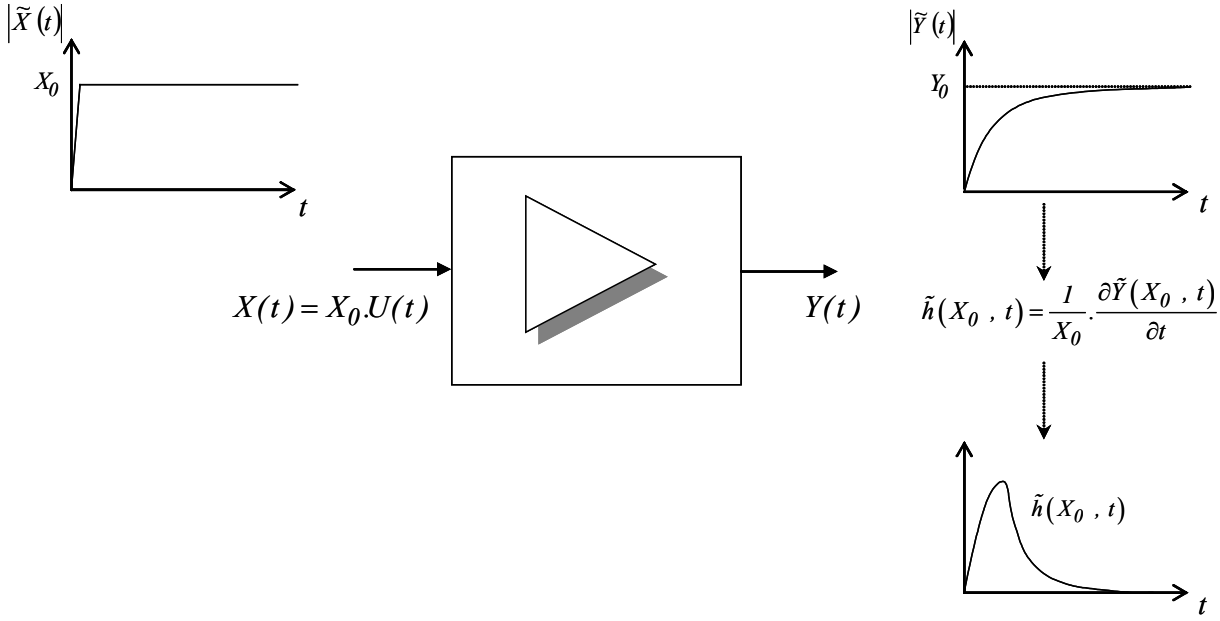


Figure I-24 : Principe d'identification de la réponse impulsionnelle non linéaire

V.4.5. Le modèle à séries de Volterra « modulées »

Afin d'améliorer la prise en compte des effets de mémoire long et court terme, un nouveau modèle a été développé par C. Mazière [I-45], basé sur une combinaison du modèle développé par N. Le Gallou et A. Soury. Ce modèle est basé sur l'observation que l'amplificateur à l'état solide peut être globalement vu comme un système bouclé représenté ci-dessous. Dans ce système bouclé, on distingue les effets de mémoire à court terme et les effets à long terme.

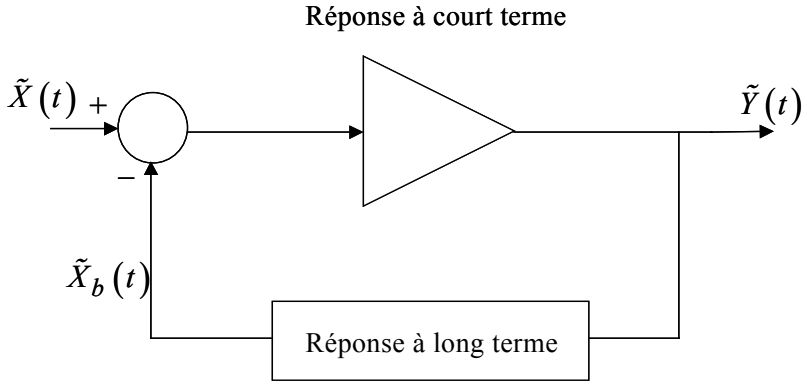


Figure I-25 : Description par système bouclé

Les effets de mémoire à court terme représentent l'amplification et le filtrage dans la bande de fonctionnement de l'amplificateur. Ils sont représentés dans la figure I-25 par la fonction de transfert directe. Lorsqu'un signal à enveloppe variable traverse cette branche du système, une partie de la sortie est réinjectée vers l'entrée suivant une dynamique non linéaire impliquant des constantes de temps longues. Cette dynamique à long terme peut être vue comme une modulation lente du point de polarisation de l'amplificateur résultant des effets thermiques, des effets de pièges et des effets de filtrage réseau de polarisation.

Considérons $\tilde{X}(t)$, $\tilde{Y}(t)$ et $\tilde{X}_b(t)$ les différents signaux de la boucle représentée sur la figure I-25. Pour décrire la fonction de transfert, on utilise la série de Volterra dynamique à un noyau, qui comme indiqué précédemment permet une bonne prise en compte de phénomènes de mémoire court terme.

$$\tilde{Y}(t) = \tilde{f}_{CT}\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t) = \frac{1}{2\pi}\int_{-BW/2}^{+BW/2} \tilde{H}_{CT}\left(\left|\tilde{X}(t)\right|,\Omega\right).X\left(\Omega\right).e^{j\Omega t}.d\Omega \qquad \text{(I-38)}$$

La boucle de contre réaction ramène à l'entrée une partie du signal de sortie suivant une dynamique à long terme. Le signal issu de cette partie peut s'écrire comme la sortie de l'amplificateur pondérée par un facteur qui dépend de son historique à long terme :

$$X_b(t) = \tilde{f}_{LT}\left(\left|\tilde{X}(t)\right|,t\right).\tilde{Y}(t) \qquad \text{(I-39)}$$

L'équation en boucle fermée s'écrit alors comme suit :

$$\tilde{Y}(t) = \frac{f_{CT}\left(\left|\tilde{X}(t)\right|,t\right).\tilde{X}(t)}{1 - f_{LT}\left(\left|\tilde{X}(t)\right|,t\right).f_{CT}\left(\left|\tilde{X}(t)\right|,t\right)} \qquad \text{(I-40)}$$

Si l'on considère que le gain de boucle $\tilde{f}_{LT}(\bullet).\tilde{f}_{CT}(\bullet)$ est faible devant l'unité, alors on obtient la forme plus simple suivante :

$$\begin{cases} \tilde{Y}(t) \cong \tilde{Y}_1(t).\tilde{Y}_2(t) \\ \tilde{Y}_1(t) = \tilde{f}_{CT}\left(\left|\tilde{X}(t)\right|,t\right).\tilde{X}(t) \\ \tilde{Y}_2(t) = 1 + \tilde{f}_{LT}\left(\left|\tilde{X}(t)\right|,t\right).\tilde{f}_{CT}\left(\left|\tilde{X}(t)\right|,t\right) \end{cases} \qquad \text{(I-41)}$$

où $\tilde{Y}_1(t)$ est la réponse de l'amplificateur court terme et $\tilde{Y}_2(t)$ est un terme de modulation qui contient à la fois les dynamiques à long et court terme.

On remarque que le terme $\tilde{Y}_2(t)$ dépend uniquement du module du signal d'entrée et non de la phase. Ce terme va être représenté par un modèle à réponse impulsionnelle pour prendre en compte la dynamique à long terme.

$$\tilde{Y}_2(t) = 1 + \int_0^{Tm} \tilde{h}_{FB}\left(\left|\tilde{X}(t-\tau)\right|, \tau\right).\left|\tilde{X}(t-\tau)\right|.d\tau \tag{I-42}$$

A partir de l'expression (I-38), (I-41) et (I-42), on peut définir le terme $\tilde{Y}_1(t)$ comme suit :

$$\tilde{Y}_1(t) = \frac{1}{2\pi} \int_{-BW/2}^{+BW/2} \tilde{H}_{CT}\left(\left|\tilde{X}(t)\right|, \Omega\right).\left|\tilde{X}(\Omega)\right|.e^{j\Omega t}.d\Omega \tag{I-43}$$

La procédure d'identification se passe en deux phases. La première phase consiste à extraire le noyau à court terme $\tilde{H}_{CT}\left(\left|\tilde{X}(t)\right|, \Omega\right)$, l'extraction de ce noyau est réalisée à l'aide d'un signal CW comme dans le cas du modèle de Volterra un noyau. La deuxième phase consiste à extraire la réponse non linéaire $\tilde{h}_{FB}\left(\left|\tilde{X}(t-\tau)\right|, \tau\right)$ à l'aide d'un échelon d'Heaviside d'amplitude variable modulant une porteuse fixe à ω_0 comme dans le cas du modèle à réponse impulsionnelle non linéaire.

VI. Conclusion

Les méthodes d'analyse des circuits micro-ondes telles que la méthode temporelle, la méthode d'équilibrage harmonique et la méthode basée sur le transitoire d'enveloppe, sont arrivées à des degrés de maturité (précision et rapidité) élevés. Par contre les simulateurs « système » ont connu, quant à eux, un développement moindre, et restent encore aujourd'hui relativement sommaires.

La complexité structurelle des systèmes de télécommunication actuels et la disparité des signaux à transmettre sont tels qu'ils représentent des obstacles importants à l'évaluation des performances. La prédiction de ces performances devient de plus en plus critique en vue d'une réalisation optimale en termes de dimensionnement, de fiabilité et de coût. Cela nécessite des simulations de type système qui soient performantes et capables de prédire les phénomènes dispersifs non linéaires rencontrés dans les dispositifs amplificateurs. C'est dans ce contexte que des efforts importants ont été conduits ces dernières années afin d'établir des techniques de macro-modélisation des amplificateurs alliant précision, stabilité et rapidité. Dernièrement des nombreux travaux sur la modélisation des phénomènes de mémoire ont été menés, en particulier les modèles qui prennent en compte les phénomènes de mémoire non linéaires dans les amplificateurs de puissance. Nous avons présenté dans la deuxième partie de ce chapitre les trois modèles qui ont été développé au sein de notre équipe. Les premiers modèles, développé est le modèle de N. Le Gallou, dont le point de départ a été les séries de Volterra limitées au premier ordre, ces trois modèles ont montré des aptitudes fortes à reproduire le comportement de circuits à mémoire.

VII. References

[I-1] L.O. Chua, and P.M. Lin

« Computer aided analysis of electronic circuit : algorithms and computational techniques », Prentice-hall, Englewood Cliffs, N J, 1975.

[I-2] J. Vlach. K. Singhal

« Computer Methods for Circuit Analysis and Design », Van Nostrand Reinhold, 2nd edition, 1994.

[I-3] G. Dahlquist and A. Bjoerck

« Numerical Methods », Prentice-hall, Englewood Cliffs, N J, 1979.

[I-4] S. Skelboe

« Computation of periodic steady state response of nonlinear networks by extrapolation methods », IEEE Transactions on Circuits and Systems , vol. CAS-27, No 3, pp.161-175, March 1980.

[I-5] T.J. Aprille and T.N. Trick

« Steady state analysis of nonlinear circuits with periodic inputs », Proceedings of the IEEE , vol. 60, No 1, pp.108-114, Jan 1972.

[I-6] E. Ngoya

« Contribution à la création d'outils de CAO des circuits non linéaires microondes » Thèse de Doctorat de l'Université de Limoges, n° d'ordre 36-88, Octobre 1988.

[I-7] V. Rizzoli and A. Neri

« State of art and present trends in non linear microwave CAD techniques », IEEE Transactions on Microwave Theory and Techniques, vol 36, No 2, pp. 343-365, February 1988.

[I-8] C. E. Christofferson, M. B. Steer and M. A. Summers

« Harmonic balance analysis for systems with circuit-field iterations », Proc. IEEE MTT-S International Microwave Symp, pp. 1131–1134, June 1998.

[I-9] M.B. Steer, J.W. Bandler, C.M. Snowden

« Computer-aided design of RF and microwave circuits and systems », IEEE Transactions on Microwave Theory and Techniques, vol. 50, No. 3, pp. 996-100 March 2002.

[I-10] Y. Saad

« Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics », Philadelphia, PA, 2003.

[I-11] E. Ngoya, J. Sombrin et J. Rousset

« Simulation des circuits et Systèmes: Méthodes Actuelles et Tendances », Séminaires Antennes Actives et MMIC, pp. 171-176, Arles (France), Avril 1994.

[I-12] E. Ngoya and R. Larcheveque

« Envelop Transient Analysis: A new method for the transient and steady-state analysis of microwave communication circuits and systems », IEEE MTT Symp. Dig., p.1365-1368, June 1996.

[I-13] E. Ngoya, R. larchevèque

« Simulation of microwave communication circuits and systems by Envelope and Compressed Transient methods », Proceedings of the GAAS and associated CAD Workshop, pp. 7A2-7A3, Paris 1996.

[I-14] J. Liu

« Continuous Time and Mixed-Signal Simulation in Ptolemy II », MS Report, UCB/ERL Memorandum M98/74, Dept. of EECS, University of California, Berkeley, CA 94720, December 1998.

[I-15] L. Muliadi

« Discrete Event Modeling in Ptolemy II », Technical Report, Dept. of EECS, University of California, Berkeley, CA 94720, May 1999.

[I-16] E. A. Lee and T. M. Parks

« Dataflow Process Networks », Proceedings of the IEEE, vol. 83, No. 5, pp. 773-801, May 1995.

[I-17] E. A. Lee and D. G. Messerschmitt

« Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing », IEEE Transactions on Computers, vol. C-36, No. 1, January 1987.

[II-18] The MathWorks Inc

« Simulink : Dynamic System Simulation for MATLAB », Using Simulink, User's Guide, version 6, 2000.

[II-19] R. Nikoukhah, S. Steer

« Scicos - A Dynamic System Builder and Simulator », User's Guide, version 1.

[I-20] R. Nikoukhah

« Scicos: a dynamic systems modeler and simulator », The 23rd IASTED International Conference on Modeling, Identification, and Control, Grindelwald, Switzerland, February 2004.

[I-21] Signal Processing WorkSystem (SPW)

User's Guide, Cadence Design Systems Inc , Alta Group, 1997.

[I-22] Noise/Com Incorporation

« Test set speeds NPR measurements », Microwaves & RF, pp.126-128, January 1994.

[I-23] L. Nguyen and V. Janicot

« Simulation Method to Extract Characteristics for Digital Wireless Communication Systems », Proceedings of the DATE 2001 on Design, automation and test in Europe, Germany , pp. 176-179, March 2001.

[I-24] K. Voelker

« Apply Error Vector Measurement in Communications Designs », Microwave and RF, pp. 143-152, December 1995.

[I-25] K. Blue and al

« Vectors Signal Analyzers for difficult Measurement on Time-Varying and complex modulated Signals », Hewlett-Packard Journal, pp. 6-59, December 1993.

[I-26] B. Lindoff and P. Malm

« BER Performance Analysis of a Direct Conversion », IEEE Transactions on Communications, Volume 50 No 5, pp. 856-865, May 2002.

[I-27] S. Haykin

« Communication Systems », 3rd edition, John Wiley & Sons Inc., ISBN 0-471-57176-8, 1994.

[I-28] V. Meghdadi

« Contribution à l'étude des charges utiles forte capacité : Modélisation de l'étage d'amplification de puissance – Méthodes de démultiplexages numériques », Thèse de Doctorat de l'Université de Limoges, n° d'ordre 49-98, Novembre 1998.

[I-29] H.B. Poza and al

« A wideband datalink computer simulation model », Proc. NAECON conf. 1975.

[I-30] W. Bösch and G. Gatti

« Measurement and simulation of memory effects in predistortion linearizers », IEEE Transactions on microwave theory and techniques, Vol 37, n° 12, pp 1885-1890, December 1989.

[I-31] N. Le Gallou

« Modélisation par séries de Volterra dynamiques des phénomènes de mémoire non linéaires pour la simulation système d'amplificateurs de puissance », Thèse de Doctorat de l'Université de Limoges, n° d'ordre 55-2001, Novembre 2001.

[I-32] A. Soury

« Modélisation des phénomènes de mémoire à long terme des amplificateurs de puissance pour la simulation des systèmes de télécommunications », Thèse de Doctorat de l'Université de Limoges, n° d'ordre 39-2002, Octobre 2002.

[I-33] C. Mazière

« Modélisation comportementale d'amplificateurs à mémoire pour des applications à large bande utilisées dans les systèmes de télécommunications et les systèmes Radars », Thèse de Doctorat de l'Université de Limoges, n° d'ordre 40-2004, Novembre 2004.

[I-34] V. Volterra

« Theory of functionals and of integral of integro-differential equations », Dover publications, Inc., N.Y. 1958.

[I-35] N. Wiener

« Nonlinear problems in random theory », The technology press, M.I.T. and John Wiley & Sons, Inc., N.Y., 1958.

[I-36] N. Le Gallou, D. Barataud, H. Buret, J.M. Nébus, E. Ngoya

« A novel measurement method for the extraction of dynamic Volterra kernels of microwave power amplifiers », European Microwave Week, GaAs Conference, Oct 2000, Paris.

[I-37] M. Asdente, M.C. Pascussi, A.M. Ricca

« Modified Volterra Wiener functional method for highly non linear systems », Alta Frequenza, Vol XLV, n° 12, pp 756-759, Dec 1976.

[I-38] F. Filicori, V.A. Monaco & Al

« Mathematical approach to electron device modelling for nonlinear microwave circuit design : state of art and present trends », Invited paper, European Transactions on Telecomunications, Vol I, n° 6, pp 641-654, Nov 1990.

[I-39] F. Filicori, G. Vanini, V.A. Monaco

« A non linear integral model of electron devices », IEEE Transactions on microwave theory and techniques, Vol 40, n° 7, pp 1456-1465, July 1992.

[I-40] N. Le Gallou, E. Ngoya, H. Buret, D. Barataud, J.M. Nébus

« An improved behavioral modeling technique for high power amplifiers with memory », IEEE MTT-S International Microwave Symposium Digest, pp 983-986, Phoenix, 2001.

[I-41] N. Le Gallou, E. Ngoya, D. Batataud, J.M. Nebus

« Behavioral modeling of power amplifiers for accurate wide band system simulation », paper 7, Workshop "Going Beyond S Parameters, 31st European Microwave conference EuMc, London, 2001.

[I-42] E. Ngoya, A. Soury

« Fundamentals of Nonlinear Behavioral Modeling for RF and Microwave Design », Artech House Publishers, May 2005.

[I-43] A. Soury and al.

« Measurement based Modeling of Power Amplifier for reliable Design of modern Communication Systems », IEEE MTT-S International Microwave Symposium Digest-CDROM , Philadelphia, June 2003.

[I-44] A. Soury, E. Ngoya, J. M. Nebus

« A new behavioral Model taking into Account nonlinear Memory Effects and Transient Behaviors in Wideband SSPAs », IEEE MTT-S International Microwave Symposium Digest, pp. 853-856, Seattle 2002.

[I-45] C. Maziere, A. Soury, E. Ngoya, J. M. Nebus

« A behavioral Model of solid State Power Amplifiers taking into Account both long Term and short Term Memory Effects », International Workshop on Electronics and System Analysis, Bilbao, Octobre 2004.

CHAPITRE II

Implémentation de modèles comportementaux dans des environnements de simulation système

I. Introduction

La simulation et la modélisation des systèmes de communication RF et micro-ondes est un sujet de recherche majeur depuis plusieurs années. La complexité structurelle de ces systèmes et la disparité des signaux à transmettre représentent de réels obstacles à l'évaluation de facteurs de mérite tels que le TEB, dont l'évaluation nécessite d'importantes ressources CPU. La prédiction de ces performances devient de plus en plus critique en vue d'une réalisation optimale en terme de dimensionnement, de fiabilité et de coût. Cela implique des simulations de type système qui soient performantes.

Le problème de la simulation de systèmes est que les modèles utilisés sont, en général, trop simplifiés à cause du coût de calcul associé à la simulation de modèles précis des divers circuits qui forment le système. Il faut alors implémenter des modèles comportementaux plus significatifs qui prennent en compte les effets de mémoire, bruit, désadaptation, cependant il faut que ces modèles maintiennent des temps de calcul faibles, compatibles avec la démarche de conception système.

C'est dans ce contexte que des efforts importants ont été conduits ces dernières années afin d'établir des techniques de macro-modélisation de circuits fonctionnels (amplificateurs, mélangeurs, etc.) alliant précision, stabilité et rapidité, avec notamment la prise en compte plus ou moins fine du comportement dispersif des dispositifs haute fréquence.

Nous nous attachons plus particulièrement dans ce chapitre à la démarche d'implémentation des modèles comportementaux dans des environnement de simulation système. Nous présentons dans la première partie le principe général des deux simulateurs utilisés dans notre travail, il s'agit des logiciels Scilab/Scicos et Matlab/Simulink. Puis, dans la deuxième partie, nous introduisons la procédure d'implémentation des modèles de Volterra à deux, à un noyau et à réponse impulsionnelle non linéaire dans des environnements de simulation système généraux.

II. L'environnement Scicos

Scicos (*Scilab Connected Object Simulator*) est une boîte à outils du logiciel libre de calcul matriciel Scilab [II-1] [II-2]. Ce simulateur est dédié à la description et à la simulation des systèmes dynamiques hybrides [II-3] incluant des parties « continues » et « événementielles ». Un système est représenté sous la forme d'un schéma-bloc [II-4], où les blocs sont reliés entre eux via leurs entrées et sorties par des liaisons véhiculant des signaux. Le schéma-bloc de la figure II-1 illustre la représentation d'un tel système dans cet environnement.

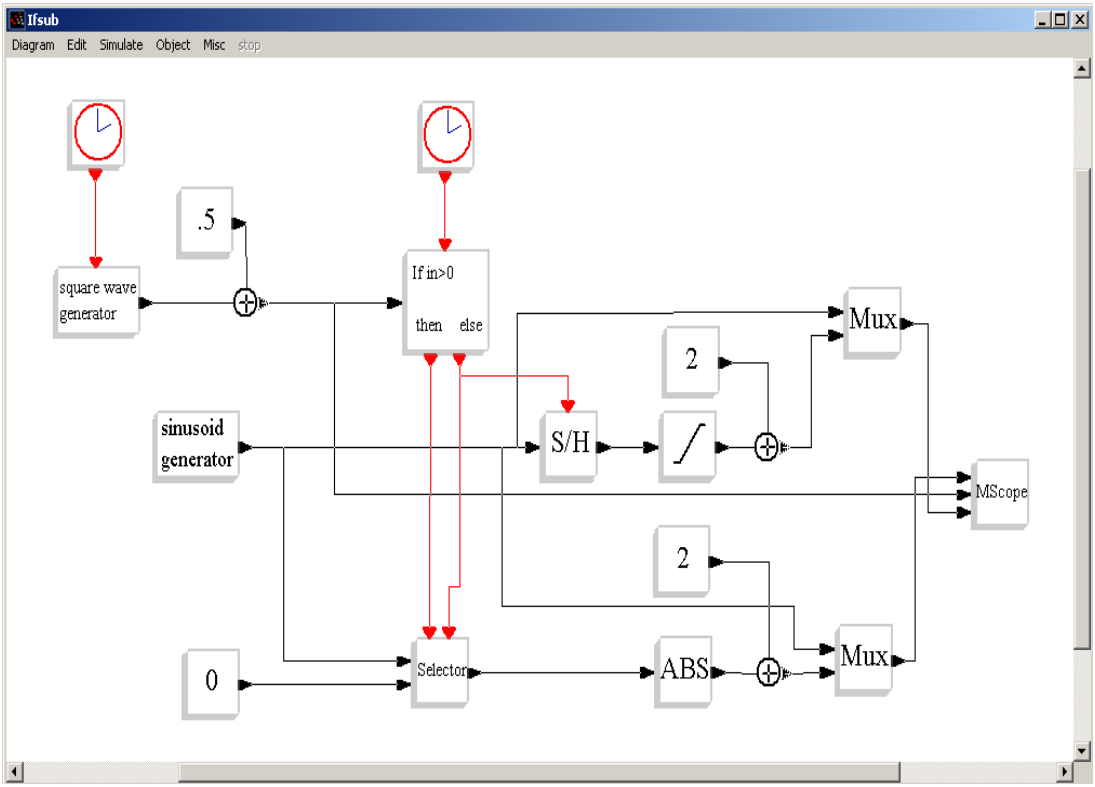


Figure II-1 : Représentation d'un système hybride sur Scicos

Les systèmes représentés sous forme de schéma-blocs peuvent être constitués d'éléments de natures différentes : continu, discret, événementiel, constituant ainsi des systèmes hybrides.

Sur l'exemple de la figure II-1, on observe un système hybride formé :

- D'une partie en temps continu (par exemple le générateur de sinusoïde) ;
- D'une partie en temps discret (par exemple le bloc *selector*).

Les types de signaux utilisés dans ce simulateur sont :

- Signaux réguliers (les liaisons sont situées à gauche et à droite des blocs) ;
- Signaux d'activation (les liaisons sont au-dessus et au-dessous des blocs).

II.1. LA STRUCTURE DE SCICOS

La figure ci-dessous représente la structure générale du logiciel Scicos qui va être décrite en détail dans ce paragraphe.

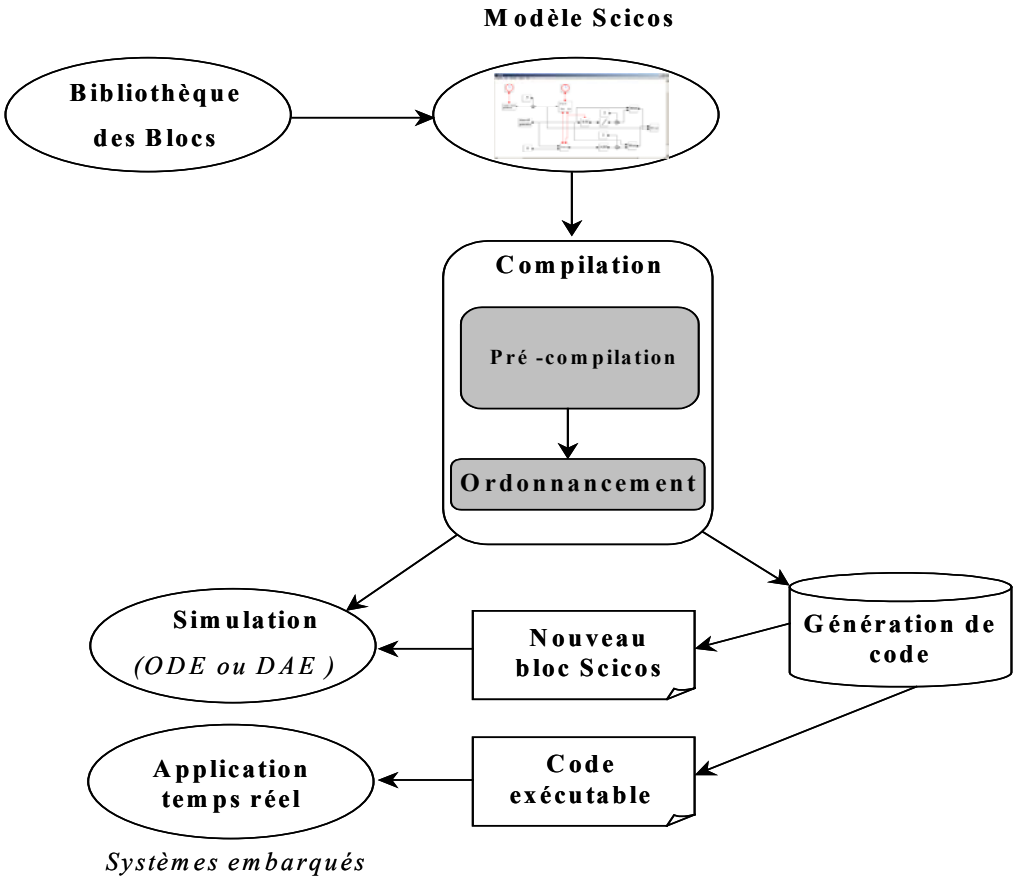


Figure II-2 : La structure de Scicos

Le logiciel Scicos comprend ainsi :

II.1.1. L'éditeur graphique

Comme nous avons vu dans le paragraphe II, Scicos fournit un éditeur graphique hiérarchique pour la construction des modèles de systèmes dynamiques en utilisant des schémas-blocs, cet éditeur est entièrement écrit en Scilab. Un nombre de blocs de base est prédéfini dans différentes palettes. Des nouveaux blocs peuvent être définis par l'utilisateur en C, Fortran ou Scilab. Une large classe de systèmes hybrides peut ainsi être modélisée.

II.1.2. Le compilateur

Le compilateur Scicos utilise le modèle de description pour déterminer l'héritage des activations des blocs et la construction des tables d'ordonnancement de la mise à jour des blocs qui seront utilisées pour la simulation et la génération de code.

II.1.3. Le simulateur

Il est de nature hybride avec interactions entre les différentes parties, discrète, continue et événementielle. Pour la partie temps-continue, selon la nature du système, on utilise le solveur LSODAR pour les équations EDO (équations différentielles ordinaires) ou le solveur DASKR pour les équations EAD (équations algébro-différentielles) [II-5].

II.1.4. Le générateur de code

Scicos peut générer du code C pour évaluer le comportement d'un sous-système utilisé par exemple dans des applications temps réel [II-6]. La simulation dans Scicos est obtenue par un algorithme décrivant le fonctionnement d'un système dynamique dans son ensemble. À cause des interactions entre les différentes parties d'un système dynamique, l'algorithme de simulation n'est pas décomposable en entités séparées. Le fonctionnement d'une partie du système ne peut pas en général être décrit par une partie distincte de l'algorithme de simulation. Grâce à la génération automatique de code, l'algorithme décrivant n'importe quelle partie du système peut être généré séparément, sous la forme d'un code en langage C.

Pour générer le code de l'algorithme de fonctionnement spécifique à un sous-ensemble de schéma bloc, il faut considérer le super-bloc indépendamment des éléments qui l'environne. Pour cela, il est nécessaire d'isoler au préalable les entrées et sorties du super-bloc et de les remplacer automatiquement par des blocs fictifs, appelés respectivement des « capteurs » et des « actionneurs », cela de manière à permettre à l'utilisateur, lors de l'implémentation du code sur une architecture matérielle, d'adapter les entrées et sorties du super-bloc à l'environnement ad hoc.

II.2. LES SIGNAUX

Comme nous l'avons présenté précédemment, les signaux utilisés pour connecter les blocs peuvent être de nature différente, réguliers ou d'activations.

II.2.1. Les signaux réguliers

Dans le schéma de la figure II.1, le générateur de sinusoïde et le générateur des signaux carrés génèrent un signal régulier par leur port de sortie régulière, le bloc MScope reçoit des signaux réguliers par ses ports d'entrée régulière. Le signal régulier est continu par morceaux, ce signal représente le résultat de calcul de la fonction qui traduit le bloc.

Notons que les signaux réguliers restent constant entre deux événements [II-7] (les blocs sont mis à jour lorsque leurs signaux d'activation sont actifs).

II.2.2. Les signaux d'activations

Les événements sont des signaux qui activent des composants de système dans les environnements entraînés par les événements discrets. Nous utilisons la notion de l'événement pour obtenir ce que nous appelons un signal d'activation qui se compose d'une union de points discrets et d'intervalles continus [II-8][II-9]. Dans l'environnement Scicos, chaque signal est alors associé à un signal d'activation, comme l'illustre l'exemple ci-dessous.

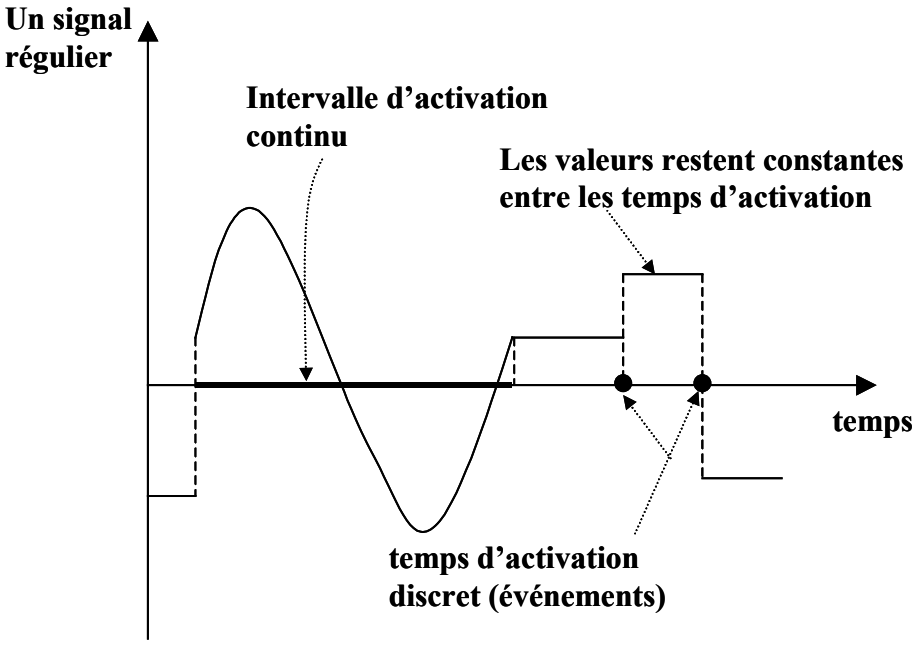


Figure II-3 : Un signal Scicos et l'ensemble des temps d'activations du bloc.

La figure II-3 montre un signal régulier en sortie d'un bloc quelconque. Ce bloc est conditionné suivant deux types d'activations : l'activation continue par intervalle et des événements ponctuels.

II.3. LES BLOCS

Les blocs de Scicos sont l'équivalent des fonctions dans un langage de programmation, certains sont disponibles dans les bibliothèques standards, d'autres peuvent être développés librement par l'utilisateur. Dans Scicos, l'utilisateur a donc le choix de pouvoir créer et utiliser des blocs personnalisés à partir des blocs existants.

On peut distinguer quatre types de blocs de base [II-10] et deux types de liaisons entre les blocs liés à la nature des ports, réguliers ou activations. Les différents types de blocs sont :

- Le bloc « *Standard* », qui représente le type de bloc le plus courant dans Scicos ;
- Le bloc « *Synchro* », utilisé le plus souvent pour le conditionnement d'autres blocs ;
- Le bloc « *Zcross* », pour des applications liées à la détection du seuil ;
- Le bloc « *Memo* », destiné à des situations de boucle algébrique.

II.3.1. Les blocs standards

Le bloc *standard* peut avoir à la fois des entrées et sorties régulières et événementielles (activations). Il peut modéliser des systèmes continus mais il lui est également possible de simuler un fonctionnement en temps discret ou encore continu et discret à la fois. Comme l'indique la figure II.4, un bloc *standard* peut être constitué de plusieurs registres, il contient des registres de sorties y_i qui stockent les signaux de sortie, il contient aussi deux registres d'état (continu x et discret z) non observables de l'extérieur du bloc.

Le bloc *standard* qui ne contient que le registre d'état discret doit être conditionné par des événements, ce bloc possède des ports d'entrée et de sortie réguliers et d'activations. A l'inverse si le bloc *standard* ne comprend que le registre d'état continu, il ne possède que des ports d'entrée et de sortie réguliers et il sera conditionné par activation continue.

Notons par u le vecteur d'entrée régulière et y le vecteur de sortie régulière, le bloc continu *standard* impose les relations (II-1) et (II-2).

$$\frac{\partial x}{\partial t} = f\left(t, x(t), z, u(t), p\right) \tag{II-1}$$

$$y(t) = h\left(t, x(t), z(t), u(t), p\right) \tag{II-2}$$

Où f et h sont les fonctions spécifiques au bloc et p est le vecteur des paramètres constants.

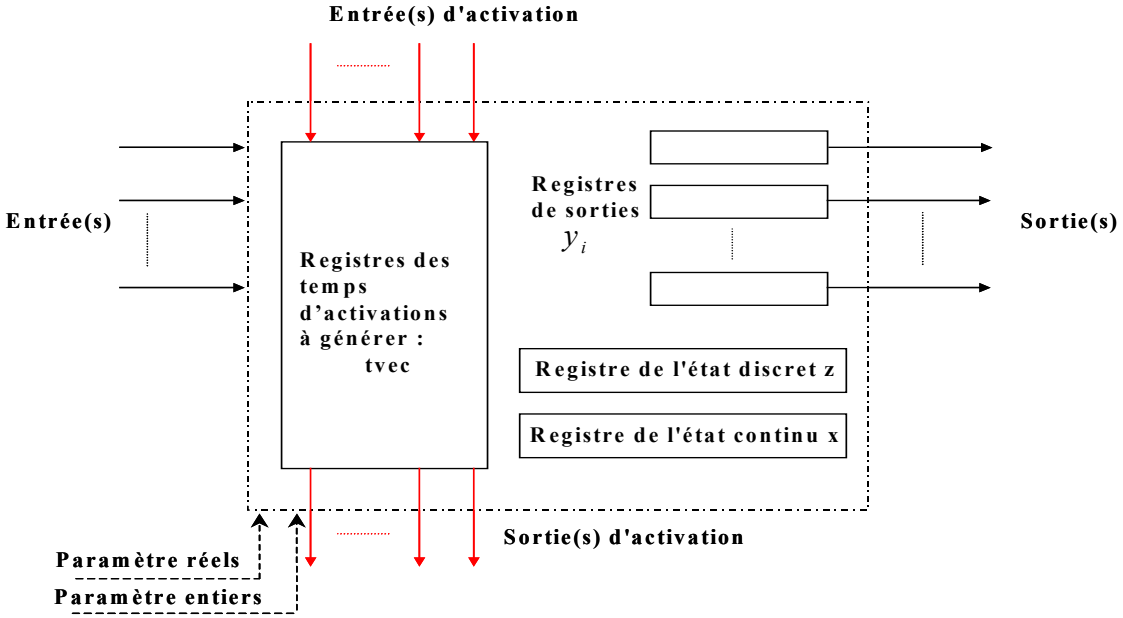


Figure II-4 : Le bloc standard

Si nous appliquons un événement au temps t_0, cet événement peut provoquer un saut des états de bloc continu *standard* dans les équations suivantes :

$$x(t_0) = g_c\left(t_0, x(t_0^-), z(t_0^-), u(t_0), p, n_0\right) \quad \text{(II-3)}$$

$$z(t_0) = g_d\left(t_0, x(t_0^-), z(t_0^-), u(t_0), p, n_0\right) \quad \text{(II-4)}$$

Où g_c et g_d sont les fonctions spécifiques du bloc, (t_0^- est le temps précédent l'occurrence de l'événement t_0), $z(t_0^-)$ est la valeur précédente de l'état discret, n_0 est un entier codant les ports activés. Si les ports d'entrée activés sont désignés par $i_1, i_2, ..., i_n$, alors :

$$n_0 = \sum_{j=1}^{n} 2^{i_j - 1} \quad \text{(II-5)}$$

Prenons par exemple un bloc comportant deux entrées d'activation. La variable n_0 peut alors prendre les valeurs suivantes :

0 : Si le bloc est activé par le bloc fictif (activation continue) ;

1 : Si le bloc est activé par le premier port d'entrée d'activations (nous commençons par la gauche du bloc) ;

2 : Si le bloc est activé par le deuxième port d'entrée d'activation (à gauche du bloc) ;

3 : Si le bloc est activé de façon synchrone, par les deux ports d'entrée d'activation.

II.3.2. Les blocs Zéro-crossing

Le bloc *Zéro-crossing* est utilisé pour détecter la traversée d'un seuil par leurs signaux d'entrées, ce bloc ne possède que des entrées régulières et des sorties événementielles (Figure II-5). Une activation de type événementielle est générée dès qu'un des signaux d'entrées du bloc change de signe. Dans ce cas, la génération de l'événement et sa synchronisation peuvent dépendre de la combinaison des entrées qui ont traversé le zéro et les signes des entrées juste avant que le passage se produise.

L'exemple le plus simple d'un bloc *Zéro-crossing* est le bloc *ZCROSS* qui se trouve dans la palette de *Treshold*. Ce bloc produit un événement si toutes les entrées traversent zéro simultanément.

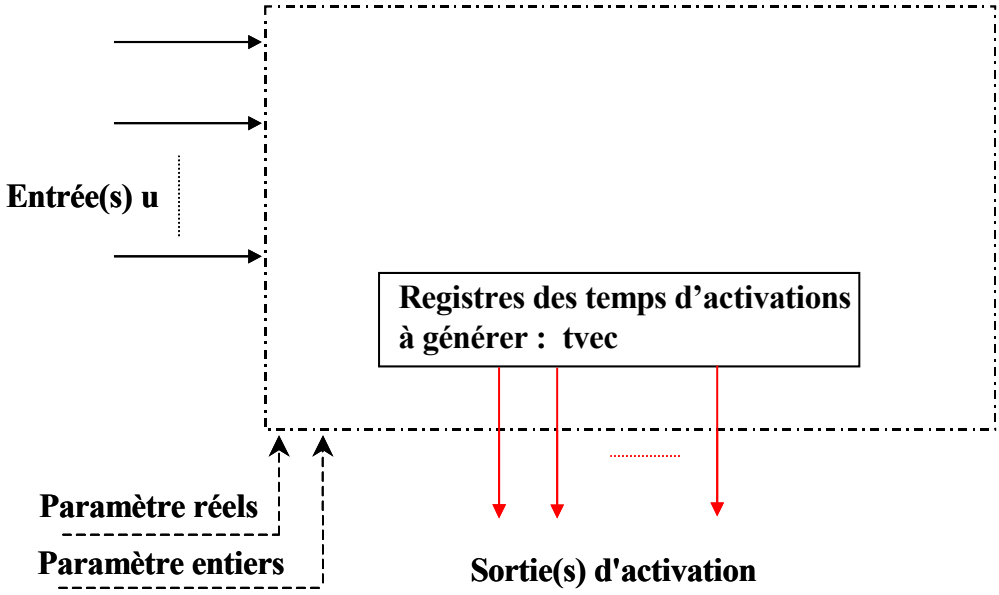


Figure II-5 : Le bloc Zéro-crossing

L'équation (II.6) indique par le vecteur de temps t_{evo}, l'instant pour lequel un évènement peut-être généré. Le vecteur t_{evo} est contenu dans le registre de sortie d'activation t_{evo} du bloc *Zéro-crossing*.

$$t_{evo} = T_z\left(t_0, u\left(t_0\right), p\right) \tag{II-6}$$

II.3.3. Les blocs Synchro

Ces blocs sont les seuls qui permettent de générer des entrées et sorties d'activations synchronisées. Ils ne possèdent qu'une seule entrée régulière et une seule entrée d'activation mais ils comptent plusieurs sorties d'activations. Suivant la valeur de l'entrée régulière et de la

fonction du bloc, l'activation ainsi générée est dirigée vers un des ports de sortie d'activation codés par un entier n_s. Cette spécificité est intéressante pour des applications de sous-échantillonnage. Le temps d'occurrence n'est pas stocké dans le registre « t_{vec} » car l'activation à générer doit être traitée immédiatement. Les valeurs de n_s sont stockées dans un registre appelé « *ntvec* ».

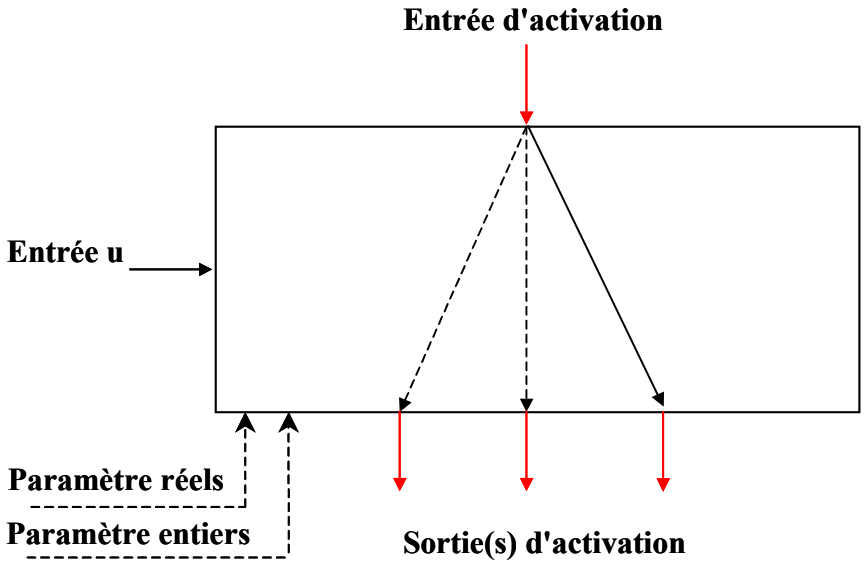


Figure II-6 : Le bloc Synchro

II.3.4. Les blocs Memo

Pour comprendre le fonctionnement des blocs *Memo*, nous avons besoin de connaître l'ordre des mises à jour des registres dans Scicos. Dans le sous-paragraphe suivant, nous allons présenter le mécanisme de mise à jour des registres dans Scicos.

II.3.4.1. Ordre de mise à jour des registres

Comme nous avons montré dans les paragraphes précédents, les blocs contiennent des différents registres suivant leur type. On distingue ainsi :

- Le registre de l'état continu ;
- Le registre de l'état discret ;
- Les registres de sorties régulières ;
- Le registre des temps d'activations à générer « t_{vec} ».

La mise à jour des différents registres peut se faire dans des ordres différents en fonction de leur type (« *régulier* », « *Zcross* », « *Synchro* » et « *Memo* ») et leur conditionnement. Cette

double dépendance détermine le fonctionnement de chaque bloc en fonction d'opérations ordonnées.

Le tableau II-1 donne le détail de ces opérations qui sont exécutées pendant la phase de simulation.

Flag	Opération
0	Mise à jour de l'état continu
1	Mise à jour des sorties
2	Mise à jour de l'état discret
3	Génération des activations (de sorties)
4	Initialisation de simulation
5	Fin de simulation
6	Ré-initialisation

Tableau II-1 : Les différents flags

Les opérations suivantes de mise à jour des registres sont exécutées systématiquement dans le même ordre préétabli :

- Le « *flag 1* » est utilisé pour la mise à jour des registres de sortie des blocs en respectant la relation d'ordre (ordonnancement) définit par le schéma bloc ;
- Le « *flag 3* » est utilisé pour la génération des événements programmés dans l'agenda de simulation par les ports de sortie d'activations ;
- Le « *flag 2* » est utilisé pour la mise à jour du registre d'état discret des blocs concernés ;
- Le « *flag 0* » est utilisé pour les blocs qui possèdent un registre d'état continu. En fait, les sorties sont mises à jour chaque fois qu'un bloc « aval » doit être mis à jour, l'intégrateur d'ODE met à jour l'état continu.

Dans le cas du bloc *Memo*, la mise à jour du registre de sortie de ce bloc correspond à celle du registre d'état discret, ce qui se traduit pour le registre de sortie par la recopie de la valeur de ses entrées au moment où les registres d'état discrets sont mis à jour. Ainsi, ce type de bloc ne peut et ne doit pas être conditionné par activation continue.

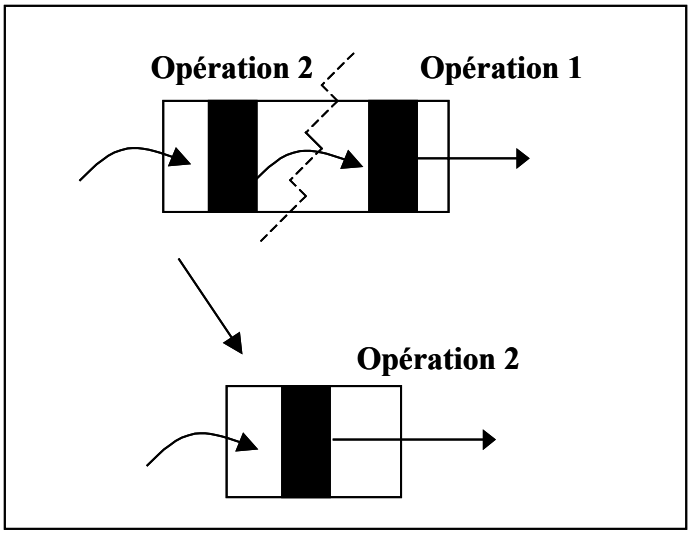


Figure II-7 : Le bloc Memo.

Opération 2 : entrée $\rightarrow$ registre d'état discret.

Opération 1 : registre d'état discret $\rightarrow$ sortie.

II.4. CONSTRUCTION D'UN NOUVEAU BLOC SCICOS

Les blocs prédéfinis dans les palettes standard de Scicos permettent de construire des schémas très divers mais, dans certains cas, une fonctionnalité décrite à partir de plusieurs blocs peut être réalisée par un seul bloc ce qui donne souvent une simulation plus efficace.
Trois types de blocs peuvent être utilisés :

- *Super Block* ;
- *Scifunc Block* ;
- *Generic Block*.

II.4.1. Super Block

Ce bloc permet de regrouper plusieurs blocs en un seul bloc et donc de construire des schémas hiérarchiques plus lisibles mais n'améliore en rien l'efficacité de la simulation car la première étape de la compilation dans Scicos consiste à mettre à plat le diagramme. En fait, un *Super Block* n'est qu'une facilité graphique pour masquer des sous-diagrammes, ce n'est pas un bloc de base.

II.4.2. Scifunc Block

Ce bloc permet d'utiliser le langage Scilab pour définir un nouveau bloc Scicos en ligne. Le principal désavantage de ce bloc est que la fonctionnalité de celui-ci est réduite. Pour pallier des manques éventuels, on fait appel à un autre type de bloc plus performant, le *Generic Block.*

II.4.3. Generic Block

Un nouveau bloc de base peut être défini de plusieurs façons, mais dans tous les cas deux fonctions sont nécessaires. Une fonction d'interface, presque toujours écrite en langage Scilab, pour gérer l'interface avec l'éditeur Scicos, et une fonction de simulation qui réalise le comportement dynamique du bloc. Cette dernière est utilisée par le simulateur et doit donc être rapide. Elle est souvent écrite en C ou Fortran, mais elle peut l'être aussi en langage Scilab, en particulier dans la phase de mise au point.

L'inconvénient d'utilisation du bloc *Generic*, comme dans le cas du bloc *Scifunc*, reste la boite de dialogue générique. Dans la plupart des cas, il est souhaitable d'adapter la fonction *GUI*, pas seulement la boite de dialogue mais également son apparence.

II.4.3.1. La fonction d'interface

La fonction d'interface d'un bloc détermine non seulement sa géométrie, sa couleur de fond, son icône, les nombres et les tailles de ses ports d'entrées-sorties, etc ..., mais aussi ses états initiaux et ses paramètres. De plus, elle gère le dialogue qui permet de modifier ses propriétés, ce que l'utilisateur peut faire en cliquant sur le bloc. Elle spécifie également le nom de la fonction de simulation utilisée par le simulateur. La syntaxe d'appel des fonctions d'interface est standard pour tous les blocs :

```
[x,y,typ]=block(job,arg1,arg2)
```

où le paramètre `job` sélectionne l'opération à réaliser :

- `job = 'plot'`: La fonction dessine le bloc.
    - `arg1` est la structure de données du bloc ;
    - `x,y,typ` sont inutilisés.

En général, nous utilisons la fonction `standard_draw` qui dessine un bloc rectangulaire ainsi que les ports d'entrées et sorties. Elle gère aussi la taille, l'icône et les couleurs du bloc.

- `job = 'getinputs'` : La fonction retourne la position et le type des ports d'entrées (régulier ou d'activation).
    - `arg1` est la structure de données du bloc ;
    - `x` est le vecteur de coordonnées x des ports d'entrées ;
    - `y` est le vecteur de coordonnées y des ports d'entrées ;
    - `typ` est le vecteur de type des ports d'entrées (1 pour régulier et 2 pour activation).

En général, nous utilisons la fonction `standard_input`.

- `job = 'getoutputs'` : Retourne la position et le type des ports de sorties (régulier ou d'activation).
    - `arg1` est la structure de données du bloc ;
    - `x` est le vecteur de coordonnées x des ports de sorties ;
    - `y` est le vecteur de coordonnées y des ports de sorties ;
    - `typ` est le vecteur de type des ports de sorties (1 pour régulier et 2 pour activation).

En général, nous utilisons la fonction `standard_output`.

- `job = 'getorigin'` : Retourne les coordonnées du plus bas point à gauche du rectangle contenant la silhouette du bloc.
    - `arg1` est la structure de données du bloc ;
    - `x` est l'abscisse ;
    - `y` est l'ordonné ;
    - `typ` est inutilisé.

    En général, nous utilisons la fonction `standard_origin`.

- `job = 'set'` : Fonction d'ouverture d'un dialogue pour l'acquisition des paramètres du bloc, s'ils existent. Ce dialogue se crée lorsque l'on clique sur le bloc considéré. Le programme doit vérifier la cohérence des paramètres fournis par l'utilisateur.
    - `arg1` est la structure de données du bloc ;
    - `x` est la nouvelle structure de données du bloc ;
    - `y` est inutilisé ;
    - `typ` est inutilisé.

- `job = 'define'` : Initialisation de la structure de données du bloc (nom de correspondance avec la fonction de calcul, le type, le nombre et la taille des ports d'entrées et sorties etc...).
    - `arg1` est inutilisé ;
    - `x` est la structure de données du bloc ;
    - `y` est inutilisé ;
    - `typ` est inutilisé.

II.4.3.2. La fonction de simulation

Cette fonction est appelée au cours de la simulation par le simulateur, la manière dont le bloc est appelé, est caractérisée par le type de la fonction de simulation.

Nous n'allons pas faire ici une présentation exhaustive de tous les types de fonctions de simulation. Les fonctions de simulation associées à la fonction de simulation de type 1 ont une liste d'appel de longueur variable en fonction des nombres d'entrées et de sorties. Par exemple, pour un bloc ayant deux entrées et une sortie, l'entête de la fonction de simulation en C sera la suivante :

```
#include "machine.h"
#include <math.h>

void mafonction(int* flag,int* nevprt,double* t,double* xd,double*
x,int* nx,double* z,int* nz,double* tvec, int* ntvec,double* rpar,int*
nrpar,int* ipar,int* nipar, double* u1,int* nu1,double* u2,int*
nu2,double* y1,int* ny1)
{
 ................
}
```

flag : Ce paramètre indique l'opération qui doit être effectuée par la fonction de simulation, il existe différentes opérations :

- **Initialisation (*flag* = 4) :** Le simulateur appelle cette fonction une fois tout au début de la simulation pour lui permettre d'initialiser ses états initiaux (cette opération est également possible par la fonction d'interface), mais également les sorties. À cette occasion, la fonction peut effectuer d'autres tâches comme par exemple ouvrir et initialiser une fenêtre graphique ou ouvrir un fichier. Elle est utilisée également pour les blocs qui exigent une allocation dynamique de mémoire.

- **Calcul des sorties (*flag* = 1) :** Le simulateur calcul les sorties en fonction des valeurs des entrées u_i et des états (x_i et z_i).
- **Mise à jour des états (*flag* = 2)** : Quand le simulateur appel cette fonction, cela signifie qu'un événement a activé le bloc (*nevprt* ≠ 0). Dans le cas où *nevprt* = -1, l'activation serait due à un événement interne de zéro-crossing. Cette fonction met à jour l'état discret du bloc en fonction de ses états courants et de ses entrées. Elle peut faire de même avec son état continu (pour modéliser des sauts éventuels).
- **Calcul de la dérivée de l'état continu (*flag* = 0):** Si le bloc contient un état continu, le solveur appelle la fonction pour calculer la dérivée de cet état.
- **Calcul des dates des événements de sortie (*flag* = 3) :** Si le bloc a des ports d'activations de sortie, la fonction fournit l'information concernant les dates des prochaines générations de ces événements.
- **Fin de simulation (*flag* = 5) :** Tous les blocs sont appelés une fois à la fin de la simulation pour qu'ils effectuent des tâches comme la fermeture des fichiers ouverts au début ou pendant la simulation, et la désallocation de mémoire si nécessaire.

Le tableau II-2 montre en détail les tâches à effectuer par chaque *flag*, les fonctions de simulations ne s'appellent pas avec toutes ces valeurs de *flag*, par exemple un bloc contenant un état discret sera appelé avec *flag* 2, et si de plus le bloc a des ports de sorties d'événement, il sera appelé aussi avec *flag* 3, mais s'il n'a aucun port de sortie, sa fonction de simulation ne s'appelle jamais avec *flag* 1. Les autres arguments de la liste d'appel de la fonction de simulation (Tableau II-2) sont des pointeurs sur les états, les entrées, les sorties, les paramètres, etc ..., et leurs dimensions. On y trouve aussi un entier (indicateur d'activation) qui spécifie dans quelle condition la fonction a été appelée (le bloc correspondant a été activé).

flag	Arguments d'entrée	Arguments de sortie	Description
1	*t,nevprt,x,z,inptr,mode,phase*	*outptr*	Mise à jour des sorties
2	*t,nevprt > 0,x,z,inptr*	*x,z*	Mise à jour des états due à l'activation externe
2	*t,nevprt = -1,x,z,inptr*	*x,z*	Mise à jour des états due à un zéro-crossing interne
3	*t,x,z,inptr,jroot*	*envout*	Programmation des activations
4	*t,x,z*	*x,z,outptr*	Initialisation des états
5	*x,z,inptr*	*x,z,outptr*	Fin de simulation
6			N'est pas nécessaire
7		*pointer,proprety*	Définition des variables algébriques et différentielles
9	*t,nevprt,x,z,inptr,phase=1*	*g,mode*	Calcul les surfaces zéro-crossing et le mode
9	*t,nevprt,x,z,inptr,phase=2*	*g*	Calcul les surfaces zéro-crossing
10	*t,nevprt,x,z,inptr,mode,phase*		Calcul de jacobien

Tableau II-2 : Les différentes valeurs de flag possibles

II.5. LES SOLVEURS

Les solveurs sont des algorithmes d'intégration numérique qui calculent la dynamique du système au cours du temps en utilisant les informations contenues dans le modèle. Scicos fournit des solveurs pour prendre en charge la simulation d'une gamme étendue de systèmes, incluant les systèmes à temps continu, discret, hybrides et multicadences de n'importe quelle taille. Ces solveurs peuvent simuler les systèmes raides et les systèmes comportant des changements d'états, tels que des discontinuités, notamment les modifications instantanées de la dynamique.

Scicos utilise deux solveurs numériques *LSODAR* et *DASKR* [II-11][II-12]. Le solveur *LSODAR* est un solveur EDO (équations différentielles ordinaires) qui est utilisé lorsque le diagramme de Scicos ne contient pas de blocs implicites. Dans ce cas la dynamique en de temps-continu du modèle s'écrit :

$$x' = f\left(t, x\right) \tag{II-7}$$

Où t représente le temps, x l'état continu ou discret, et x' la dérivée de x. Cette équation différentielle ordinaire est résolue en appelant le solveur ODE *(LSODAR)* à plusieurs reprises. A chaque appel, le solveur calcule la solution pour une période donnée. Ce système dynamique représente un *système explicite*, c'est-à-dire que les dérivées $y'(t)$ peuvent s'écrire explicitement en fonction de $y(t)$ et de t. Ce n'est malheureusement pas toujours le cas et alors les problèmes deviennent très difficiles voire impossibles à résoudre. C'est là qu'intervient le deuxième solveur de Scilab appelé *DASKR* ; il permet de résoudre une grande classe de problèmes implicites. Les deux solveurs *LSODAR* et *DASKR* sont des solveurs à pas variable, ce qui signifie qu'ils changent leurs pas pendant la simulation.

II.6. LA SIMULATION PARAMETRIQUE

Pour évaluer des facteurs de mérite tels que l'ACPR, C/I_3, TEB pour chaque niveau d'entrée, il faut être capable de simuler le diagramme Scicos à chaque niveau. Dans Scicos nous avons cette possibilité à l'aide de la fonction *scicos_simulate*, cette fonction permet de simuler le diagramme Scicos à partir de Scilab.

La séquence d'appel à exécuter dans Scilab s'écrit comme suit :

[Info]=**scicos_simulate**(scs_m,Info,%scicos_context) ;

où

- scs_m : Le nom du diagramme Scicos (.cos) ;
- Info : Une liste, ce doit être list() au premier appel, puis pour les prochains appels il faut utiliser Info de sortie en tant que Info d'entrée. Cette liste contient l'information de compilation et de simulation (elle est utilisée pour éviter la recompilation). Normalement elle ne devrait pas être modifiée.
- %scicos_context : Une structure de Scilab contenant les valeurs des variables symboliques utilisées dans le contexte et les blocs de Scicos. Cette fonctionnalité permet de changer la valeur de la variable dans la liste de contexte pour réaliser une simulation paramétrique.

III. L'environnement Simulink

Simulink est un logiciel qui permet de modéliser, simuler et analyser des systèmes dynamiques (système dont les sorties et les états évoluent au cours du temps). Ce logiciel a été produit par la société « *The MathWorks Inc* » [II-13]. C'est un outil totalement intégré au noyau de calcul de Matlab qui procure un environnement de modélisation basé sur des schémas-blocs. Il possède aussi un environnement de simulation dynamique, principalement destiné aux systèmes qui peuvent être décrits à l'aide d'équations différentielles. Simulink possède une interface graphique qui facilite l'analyse de systèmes dans le domaine temporel et fréquentiel. Cette approche est en effet facilitée dans le sens où les systèmes ne sont plus décrits par des lignes de codes Matlab mais simplement définis par des schémas-blocs dont tous les éléments sont prédéfinis dans des bibliothèques de blocs élémentaires qu'il suffit d'assembler. Le schéma-bloc de la figure II-8 illustre la représentation d'un tel système.

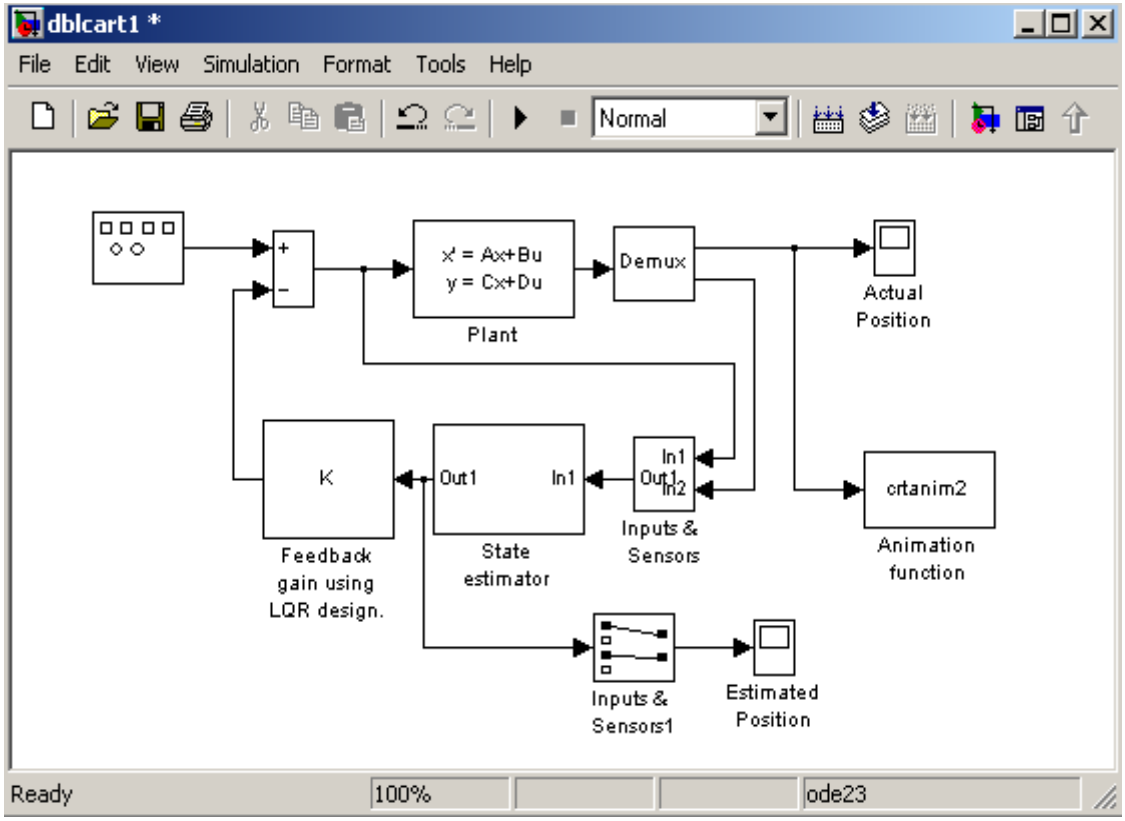


Figure II-8 : Représentation d'un système hybride sur Simulink

Le système modélisé sous Simulink peut recevoir des données de l'espace de travail de Matlab ou y envoyer des données de sortie. L'échange de données entre Simulink et l'espace de travail Matlab peut se faire à l'aide de variables communes ou par l'intermédiaire de fichiers MAT.

Au niveau de modélisation, Simulink met à disposition :

- Un éditeur graphique, dans lequel les blocs sont disposés et reliés entre eux avec des fils de connexion (Figure II-8) ;
- Des bibliothèques de blocs paramétrables .

NB : Simulink permet l'édition de gros programmes de simulation, car il ne définit pas de limite de taille de modèle, ni de limite de niveau de hiérarchie.

Au niveau de simulation, Simulink permet :

- La simulation de systèmes en temps continu (dont on connaît la valeur à chaque instant) ; discret (mono ou multi-cadencé, par opposition à continu) ou hybride ;
- La simulation de systèmes linéaires (défini sous forme matricielle) et non-linéaires ;
- La simulation dynamique avec des algorithmes puissants (Gears, Adams, Runge-Kutta,…).

Au niveau de l'analyse, Simulink permet :

- La simulation interactive, il est possible d'utiliser Simulink comme interface utilisateur et de modifier les paramètres des blocs en cours de simulation ;
- Une mise au point aisée, notamment grâce aux scopes qui permettent une visualisation dynamique des résultats en cours de simulation.

NB : Couplé à la ToolBox «Real Time Workshop», Simulink permet également l'analyse de processus en temps réel.

III.1. MODELISATION DES SYSTEMES DYNAMIQUES

Un schéma-bloc de Simulink est une représentation graphique d'un modèle mathématique d'un système dynamique. Ce modèle est décrit par un ensemble d'équations, algébriques, différentielles, et de différence.

III.1.1. Les états

Deux types d'états peuvent se produire dans un modèle de Simulink : états discrets et continus. Un état continu change sans interruption, un état discret est une approximation d'un état continu où l'état est mis à jour en utilisant des intervalles (périodiques ou apériodiques) finis.

III.1.2. Temps

La détermination du comportement d'un système avec le temps nécessite d'exécuter à plusieurs reprises le modèle à des intervalles appelés temps d'échantillonnage, depuis le début et jusqu'à la fin de la période de simulation. Simulink se rapporte à l'exécution répétée d'un modèle aux temps d'échantillonnage successifs en tant que simulation du système que le modèle représente. Il est également possible de simuler un système manuellement ; cependant, le moteur de Simulink accomplit cette tâche automatiquement sur commande de l'utilisateur.

III.1.3. Systèmes et sous-ensembles

Un schéma fonctionnel de Simulink peut être construit par couches, chacune définissant alors un sous-ensemble. Un sous-ensemble fait partie du schéma fonctionnel global et n'a idéalement aucun impact sur la signification du schéma fonctionnel. Des sous-ensembles sont fournis principalement pour aider dans les aspects d'organisation du schéma fonctionnel, ils ne définissent donc pas un schéma fonctionnel séparé.

Simulink différencie deux types de sous-ensembles: virtuel et non-virtuel. La différence principale est que les sous-ensembles non-virtuel donnent la possibilité de contrôler quand le contenu du sous-ensemble est évalué.

III.1.4. Signaux

Simulink utilise le terme *signal* pour se rapporter à une quantité variable du temps qui possède une valeur à chaque point de l'axe de temps. Simulink nous permet d'indiquer un choix de nombre entier d'attributs de signal, y compris le nom de signal, le type de données (par exemple, de 8 bits, de 16 bits, ou de 32 bits), le type numérique (réel ou complexe), et la dimensionnalité (unidimensionnelle ou bidimensionnelle).

III.2. CONSTRUCTION DES NOUVEAUX BLOCS DANS SIMULINK

Quand nous travaillons dans un domaine particulier, nous manipulons des objets récurrents et qui ne font pas obligatoirement partie des éléments de base. A cet effet, il est intéressant de personnaliser Simulink par ses propres bibliothèques de modèles.

La première solution est de passer par les sous-systèmes. On crée un schéma Simulink qui réalise une fonction particulière ou qui représente un système souvent utilisé et on groupe tous

les éléments constitutifs de ce système en un seul bloc de type sous-système. Celui-ci pourra alors être enregistré comme un nouveau composant de Simulink et être incorporé à l'une des bibliothèques préexistantes.

La deuxième solution est d'utiliser les blocs qui s'appellent S-Functions (System-Functions) [II-14]. Une S-Function est un langage de programmation d'un bloc Simulink. Les S-Functions peuvent être écrites en langage Matlab, C++, C, ADA ou Fortran et elles sont compilées comme des dossiers Matlab *External files* en utilisant l'utilitaire MEX. La S-Function est ensuite incorporée dynamiquement à Matlab, une fois nécessaire. Si on programme en langage Matlab, la S-Function écrite sera un M-file, sinon, ce sera un MEX-file. Dans tous les cas, le programme écrit est directement utilisable sous Simulink via le bloc de type S-Function que l'on trouve dans les bibliothèques de Simulink. Ce bloc fait l'interface entre Simulink et Matlab et peut être utilisé comme n'importe quel autre bloc Simulink au sein d'un schéma.

III.2.1. Les S-Functions

Une S-Function est formée de deux objets indissociables : un bloc Simulink de type S-Function et un programme [II-15]. Le bloc Simulink est l'objet qui va présenter le système spécifique que l'on a voulu ajouter à Simulink. Le programme décrit le comportement dynamique de ce système. Lors d'une simulation, Simulink va dialoguer avec le programme par l'intermédiaire du bloc S-Function pour aller chercher les éléments nécessaires à l'évaluation de la sortie du bloc S-Function.

Les S-Functions emploient une syntaxe d'appel spéciale qui permet d'interagir avec Simulink. La forme d'une S-Function est très générale, elle s'adapte facilement aux systèmes continus, discrets ou hybrides.

Pour simplifier la structure du programme décrivant la S-Function et ainsi pouvoir appréhender dans un cadre imposé le plus grand nombre possible de modèles, tout système (qu'il soit continu, discret, hybride, monovariables ou multivariables) doit être décrit de façon à entrer dans le cadre suivant :

Le bloc S-Function doit être vu comme un vecteur d'entrée u et un vecteur de sortie y (Figure II-9) et dont les caractéristiques internes sont gouvernées par l'évolution de son vecteur d'état $x = \begin{bmatrix} x_c \\ x_d \end{bmatrix}$ où x_c est la partie continue du vecteur d'état à l'instant courant et x_d la partie

discrète du vecteur d'état à l'instant courant. La sortie à l'instant courant est fonction de x, de u et du temps $y = f_0(t, x, u)$.

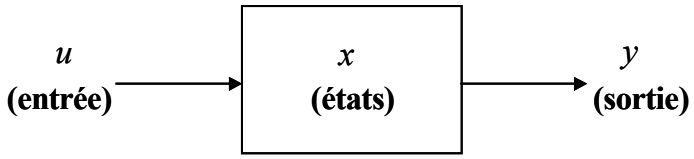


Figure II-9 : Un bloc Simulink

L'évolution du vecteur d'état x est gouvernée par une équation différentielle pour sa partie continue et une équation récurrente pour sa partie discrète.

III.2.1.1. Les étapes de simulation

L'exécution d'un modèle Simulink procède par plusieurs étapes :

- Dans un premier temps, le modèle est initialisé. Simulink incorpore le bloc librairie au modèle, permettant d'ajouter des paramètres divers, la période d'échantillonnage, types de donnée, l'ordre d'excusions du bloc et l'allocation de mémoire ;
- Ensuite, Simulink entre dans une boucle de Simulation où chaque changement d'état se réfère à un nouveau pas ;
- Pendant chaque étape de simulation, Simulink exécute chacun des blocs du modèle dans l'ordre déterminé lors de l'initialisation. Pour chaque bloc, Simulink appelle les fonctions qui calculent les états, les dérivées, et les sorties du bloc pour chaque temps d'échantillonnage, ceci jusqu'à ce que la simulation soit complète.

La figure II-10 illustre les différentes étapes de simulation.

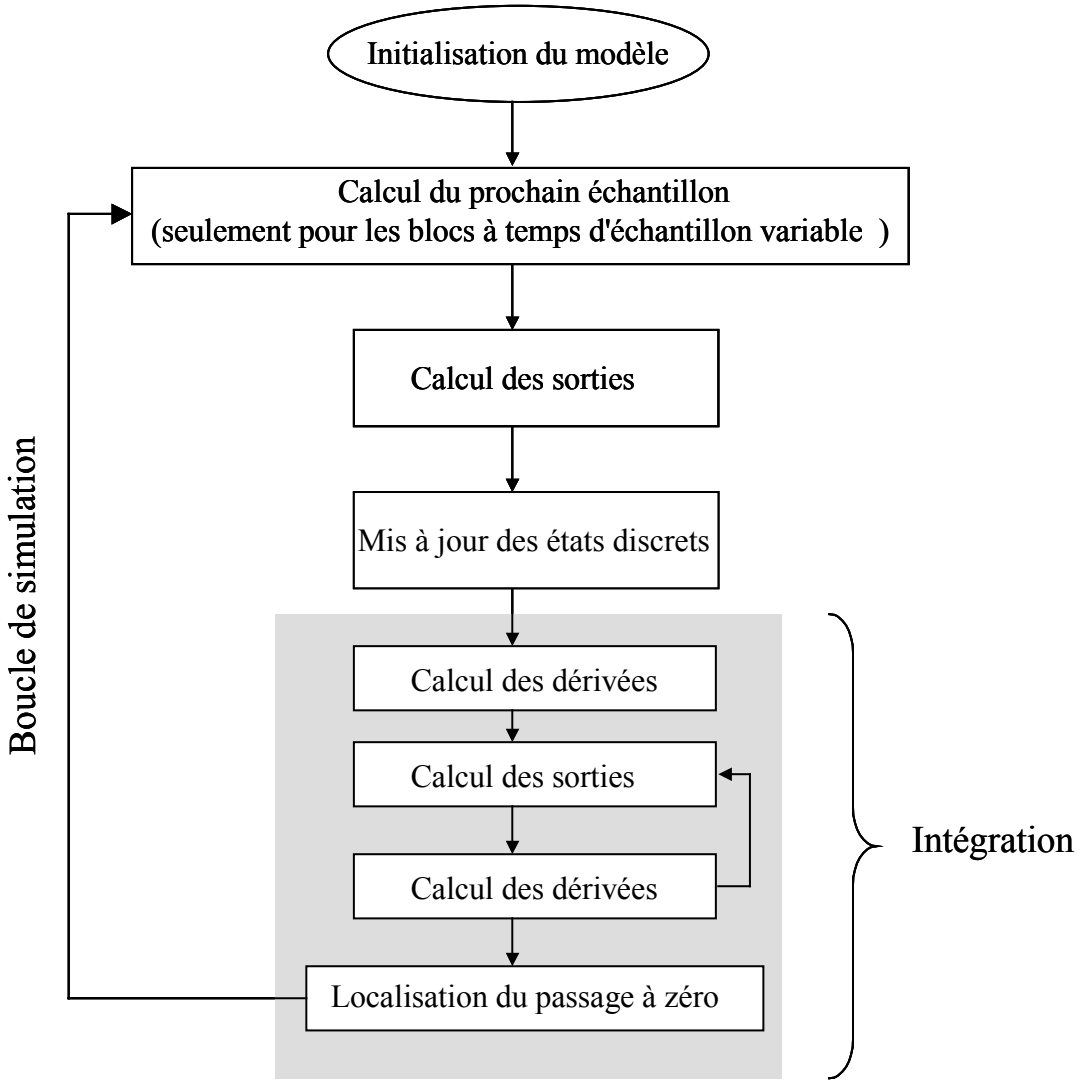


Figure II-10 : Etapes de simulation

III.2.2. Les routines de S-Functions

Une S-Function possède un ensemble des routines qui exécutent les tâches requises à chaque étape de simulation [II-16]. Lors de la simulation d'un modèle, à chaque étape, Simulink appelle les routines appropriées pour chaque bloc S-Function.

Les routines qui sont utilisées dans la plupart des blocs sont décrites ci-dessous :

- *mdlInitializeSizes*

Cette fonction permet de définir le nombre d'entrées et de sorties, la taille des ports, la période d'échantillonnage, les options à inclure au projet, mais également tout autre objet, comme le nombre d'états requis par la S-Function.

- *mdlInitializeSampleTimes*

Simulink appelle cette fonction pour déterminer les périodes d'échantillonnage de la S-Function.

- *mdlOutputs* :

Simulink appelle cette fonction lorsque le calcul de la sortie est nécessaire. Cette fonction utilise une macro de *Simstruct* pour accéder au signal d'entrée par pointeur (uPtrs). La macro renvoie un vecteur de pointeur, accessible par la commande *uPtrs[i]. Ensuite, le programme emploie la macro real_T pour accéder au signal de sortie. La fonction int_T permet d'obtenir la largeur du signal. La S-Function fait au final une boucle sur les entrées pour calculer les sorties.

- *mdlTerminate*

C'est une routine obligatoire de la S-Function, elle permet d'accomplir les tâches à la fin de la simulation.

III.2.3. Implémentation de S-Function

Nous pouvons implémenter une S-Fonction en tant qu'une M-file ou une MEX-file. Les paragraphes suivants décrivent ces méthodes d'implémentation.

III.2.3.1. M-File S-Functions

Une M-file S-Fonction se compose d'une fonction de Matlab de la forme suivante:

$$[sys, x_0, str, t_s] = f(t, x, u, flag, p_1, p_2, ...) \quad \text{(II-8)}$$

où f est le nom de la S-Fonction, t le temps courant, x le vecteur d'état du bloc correspondant, u les entrées du bloc, *flag* indique une tâche qui doit être exécuté, et p_i les paramètres du bloc. Cette syntaxe générique permet de représenter la simulation de systèmes continus, discrets, hybrides, etc.

Pendant la simulation d'un modèle, Simulink appelle à plusieurs reprises f en utilisant le *flag* pour indiquer la tâche à exécuter. Chaque fois que la S-Fonction exécute une tâche, elle renvoie le résultat dans une structure.

Simulink propose un exemple type de M-file S-Fonction. On le trouve dans le répertoire *matlabroot*/toolbox/simulink/blocks, sous le nom de sfuntmpl.m.

Le paramètre *flag* peut prendre six valeurs différentes, ce paramètre contrôle les types de données retournées par la M-file S-Fonction dans la variables *sys* (Tableau II-3).

Etape de simulation	Routine de S-Function	flag
Initialisation	mdlInitializeSizes	flag = 0
Calcul du prochain échantillon (seulement pour les blocs à temps d'échantillon variable)	mdlGetTimeOfNextVarHit	flag = 4
Calcul des sorties	mdlOutputs	flag = 3
Mis à jour des états discrets	mdlUpdate	flag = 2
Calcul des dérivées	mdlDerivatives	flag = 1
Terminaison	mdlTerminate	flag = 9

Tableau II-3 : Les différentes valeurs de flag

III.2.3.2. MEX-File S-Functions

Comme une M-file S-Fonction, une fonction MEX-file se compose d'un ensemble de routines que Simulink appelle pour accomplir de diverses tâches pendant la simulation. Les fonctions MEX-file sont implémentées dans des différents langages de programmation : C, C++, ADA, ou Fortran. En outre, Simulink appelle les routines de MEX-file S-Fonction directement au lieu de passer par les valeurs de *flag* comme avec les M-file S-Fonctions. Le nombre de routines implémentées dans les MEX-file S-Function est donc beaucoup plus grand que dans les M-file S-Function. La fonction MEX-file a l'accès direct à la structure de données interne (*SimStruct*), Simulink utilise cette structure pour maintenir les informations sur S-Fonction. Les fonctions de MEX-file peuvent également utiliser MATLAB MEX-file API pour accéder directement à la zone de travail de MATLAB.

Si l'on désire créer un bloc relativement simple il est possible d'utiliser une méthode de création automatique sous Simulink. On utilise pour cela un bloc S-Functions Builder.
La syntaxe de MEX-file S-Fonction écrite en langage C++ est :

```
#define S_FUNCTION_NAME   nom_S-Function        // nom de la S-Fonction
#define S_FUNCTION_LEVEL 2      // fonctionne avec la version 2.2 et ultérieur
#include "simstruc.h"   // fichier d'en-tête qui définit la structure
Simulink et les macros d'accès de Simstruct

static void mdlInitializeSizes(SimStruct *S)
{
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
}
static void mdlTerminate(SimStruct *S)
{
}
```

```
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

La première ligne contient le nom de la S-Function, celui même qui sera utilisé quand on emploiera une S-Function dans un modèle Simulink/RTW. La seconde ligne indique le format de la S-Function (Format 2 pour Simulink 2.2 et supérieur). Pour être compilée correctement, le début de la S-Function doit contenir la structure *simstruct*.

III.3. SOLVEURS SOUS SIMULINK

MathWorks propose une série de solveurs [II-17] [II-18] paramétrables pour résoudre des systèmes d'équations différentielles. Ces solveurs sont différents entre Matlab et Simulink, même s'ils sont proposés sous la même dénomination.

Sous Simulink, le choix du solveur est plus large. Nous avons accès à deux catégories bien distinctes : des solveurs à pas variable et des solveurs à pas fixe. Sept solveurs à pas variable sont proposés et cinq solveurs à pas variable. Simulink fournit aussi deux solveurs discrets : un solveur discret à pas fixe et un autre solveur discret à pas variable.

III.4. LA SIMULATION PARAMETRIQUE

De même que le logiciel Scicos, Simulink présente la fonctionnalité de réaliser des simulations paramétriques sous Simulink. Pour cela, il faut appeler Simulink à partir de Matlab à l'aide de la fonction *sim*.
La séquence d'appel à exécuter dans Matlab s'écrit comme suit :

[**t**,**x**,**y**] = **sim**(model,timespan,options,ut) ;

Où

- **t** : Renvoie le vecteur du temps de la simulation ;
- **x** : Renvoie la matrice d'état de la simulation (des états continus ou états discrets) ;
- **y** : Renvoie la matrice de sortie de la simulation ;
- model : Le nom du diagramme Simulik.(.mdl) ;
- timespan : Temps de début et d'arrêt de simulation.

IV. Implémentation numérique des modèles de Volterra dans un environnement système

L'implémentation numérique des modèles à mémoire, qu'ils soient unilatéraux ou bilatéraux, dans des environnements de simulation système nécessite une attention particulière sur le coût et la précision de calcul ainsi que sur la stabilité du système d'équations. Afin d'implémenter des modèles précis, rapides et stables, nous avons étudié des outils mathématiques, permettant de représenter aux mieux aussi bien les effets dispersifs (variation avec la fréquence) que les effets non linéaires (variation avec la puissance).

Dans cette partie, nous allons présenter les deux approches adoptées pour implémenter les modèles d'amplificateur dans les deux logiciels de simulation Scicos et Simulink. L'utilisation de ces deux approches nous permet d'éliminer l'intégrale de convolution et d'exprimer ainsi les réponses sous la forme d'équations différentielles.

IV.1. INTEGRATION NUMERIQUE

Soit l'intégrale suivant à évaluer :

$$I = \int_a^b f(x)dx \tag{II-9}$$

L'intégration numérique consiste à estimer la valeur de I en évaluant $f(x)$ en un nombre fini de points x_i dans l'intervalle [a,b] et en effectuant une somme pondérée des valeurs $f(x_i)$.

IV.1.1. Méthode des rectangles

Soit la fonction $f(x)$ à intégrer sur l'intervalle [a,b]. Nous découpons cet intervalle en N sous-intervalles égaux et posons :

$$x_0 = a,\ x_1 = a+h,\ x_2 = a+2h,.....,x_{n-1} = a+(n-1)h,\ x_n = a+nh$$

où $h = \frac{b-a}{n}$ représente le pas de la subdivision.

La méthode des rectangles consiste à calculer l'intégrale I par la somme des aires des n rectangles de base $[x_{i-1}, x_i]$ et de hauteur $f(x_{i-1})$, i variant de 0 à $n-1$.

L'idée de cette méthode est de faire l'approximation sur chaque intervalle, on remplace l'aire sous la courbe par une aire plus simple à calculer.

On considère sur chaque intervalle $[x_{i-1}, x_i]$, deux rectangles de largeur $x_{i+1} - x_i$. La méthode des rectangles consiste à remplacer « l'aire sous la courbe » par la somme des aires des rectangles à gauche (ou à droite) (Figure II-11).

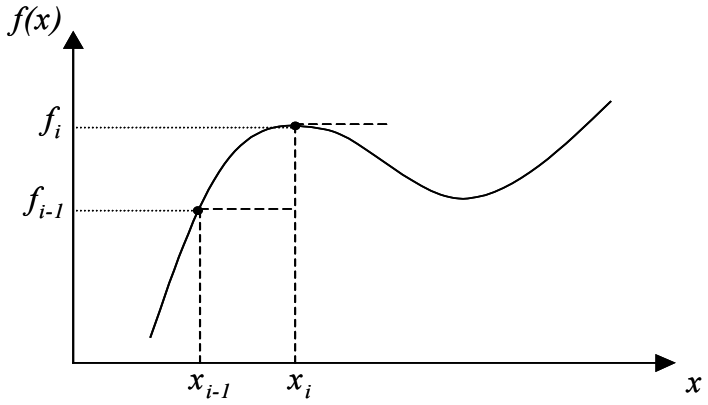


Figure II-11 : Interprétation géométrique de la méthode des rectangles à gauche

La forme générale de cette méthode est donnée par l'équation II-10 :

$$\int_a^b f(x)dx = h\sum_{i=0}^{n-1} f_i \qquad \text{(II-10)}$$

IV.1.2. Méthode des trapèzes

Cette méthode consiste à remplacer la courbe *f(x)* par une « ligne brisée » et à calculer l'aire de chaque trapèze, on obtient l'approximation de *I* par la somme des aires sur l'intervalle défini, comme l'illustre la figure suivante.

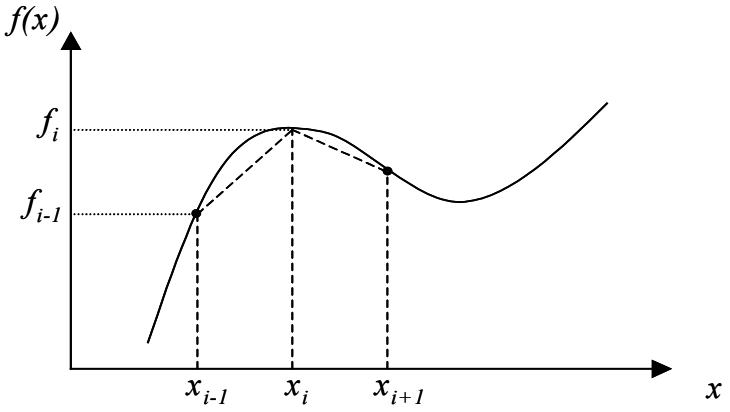


Figure II-12 : Interprétation géométrique de la méthode des trapèzes

Dans ce cas, on a :

$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{f_{i-1}+f_i}{2}.\Delta x_i \qquad \text{(II-11)}$$

où $\Delta x_i = x_i - x_{i-1}$, si on prend [a,b] comme intervalle d'intégration de *f(x)*, on a :

$$\int_{x_{i-1}}^{x_i} f(x)dx = \frac{1}{2}\sum_{i=1}^{n}\left(f_{i-1}+f_i\right).\Delta x_i \qquad \text{(II-12)}$$

Ainsi l'intégrale précédente devient :

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(a)+f(b)+2.\sum_{i=1}^{n-1} f_i \right) \qquad \text{(II-13)}$$

IV.1.3. Méthode de Simpson

Cette méthode consiste à remplacer, l'espace entre x_{i-1} et x_{i+1} par l'arc de parabole passant par f_{i-1}, f_i, f_{i+1}.

Notons x_{i-1}, x_i et x_{i+1} trois points consécutifs de la courbe représentative de la fonction *f(x)*. Le calcul des paramètres de la parabole d'équation $f = mx^2 + px + q$ passant par ces points conduira à :

$$\begin{cases} f_{i-1} = mx_{i-1}^{\ 2} + px_{i-1} + q \\ f_i = mx_i^{\ 2} + px_i + q \\ f_{i+1} = mx_{i+1}^{\ 2} + px_{i+1} + q \end{cases} \qquad \text{(II-14)}$$

L'intégrale de la fonction $mx^2 + px + q$ entre x_{i-1} et x_{i+1} peut s'écrire :

$$J = \left[2m\left(x^2_{i+1} + x_{i+1}x_{i-1} + x^2_{i-1} + 3p\left(x_{i-1}+x_{i+1}\right) + 6q \right) \right].\frac{h}{3} \qquad \text{(II-15)}$$

Compte tenu du système ci-dessus, on fait apparaître f_{i-1}, f_i, f_{i+1} dans l'expression II-15 et en considérant que $x_{i-1} = x_i - h$ et $x_{i+1} = x_i + h$, on obtient :

$$J = \left(f_{i-1} + 4f_i + f_{i+1}\right).\frac{h}{3} \qquad \text{(II-16)}$$

En conséquence, pour une intégration sur l'intervalle [a, b], en regroupant les points de la subdivision trois par trois en commençant à $x_o = a$: (x_o, x_1, x_2) , (x_2, x_3, x_4) , (x_4, x_5, x_6), etc., on remarque que le nombre n de points de la subdivision doit être pair et en utilisant la relation de Chasles [II-19] pour les intégrales, on obtient la formule d'approximation de Simpson :

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(a) + f(b) + 4.\sum_{\substack{i=1 \\ i \to impair}}^{n-1} f_i + 2.\sum_{\substack{i=2 \\ i \to pair}}^{n-2} f_i \right) \qquad \text{(II-17)}$$

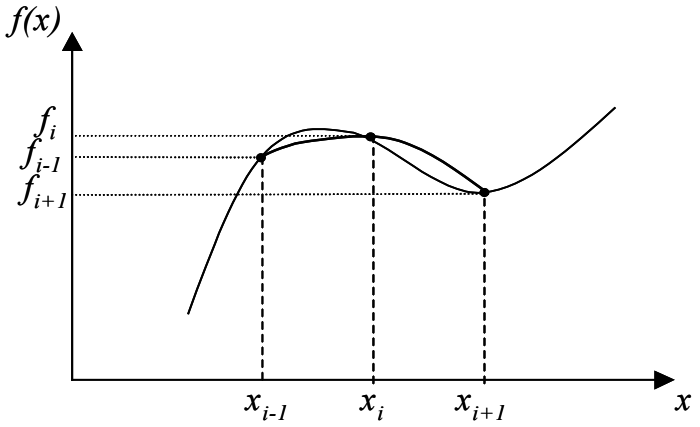


Figure II-13 : Interprétation géométrique de la méthode de Simpson

IV.1.4. Formules de Newton-Cotes

Les méthodes des rectangles, des trapèzes et de Simpson sont des cas particuliers des formules de Newton-Cotes.

L'intégrale *I* de l'équation (II-9) est une combinaison linéaire de $f(x_i)$, ce qui signifie que cette intégrale est calculée de façon exacte lorsque $f(x)$ est un polynôme de degré inférieur ou égal à *n.* Si on considère que le nombre de valeurs x_i est égal à $n+1$, on intègre sur *n* tranches, et les valeurs des coefficients de la combinaison linéaire dépendent de *n*. Les résultats de cette méthode donnent :

$$\int_{x_0}^{x_n} f(x).dx \approx \alpha_0 f(x_0) + \alpha_1 f(x_1) + ... + \alpha_k f(x_k) + ... + \alpha_n f(x_n) \qquad \text{(II-18)}$$

Soit encore :

$$\int_{x_0}^{x_n} f(x).dx = \frac{h}{A}\sum_{k=0}^{n'} a_k f\left(x_0 + k\Delta x\right) \; avec \; n' \neq n \qquad \text{(II-19)}$$

où *A* et a_k sont données par le tableau suivant :

Degré *n*	Nom de la formules	*A*	a_0	a_1	a_2
0	Rectangles	1	1		
1	Trapèzes	2	1	1	
2	Simpson	3	1	4	1

Tableau II-4 : Coefficients des formules de Newton-Côtes

Les méthodes d'intégration numérique que nous venons de voir, sont très utilisées, et elles donnent satisfaction quant à la précision des résultats.

Cependant, le temps de calcul de ces méthodes peut rapidement poser problème. En effet, le temps de calcul et la précision de simulation sont liés au rapport entre la durée de mémoire du dispositif, les constantes de temps du dispositif et les variations temporelles du signal d'entrée. De plus, l'équation des modèles de Volterra, constituée de l'intégrale de convolution, requière la connaissance simultanée de la forme d'onde et du spectre de signal d'entrée. Cette particularité nécessite la connaissance du le signal d'entrée sur une période entière afin d'évaluer l'équation du modèle par des techniques de transformation de Fourier Rapide (FFT). L'évaluation brutale de l'équation des modèles de Volterra rend les modèles pratiquement inutilisables lorsque le nombre d'échantillons temporels devient très important.

IV.2. IMPLEMENTATION DU MODELE DE VOLTERRA DEUX NOYAUX

IV.2.1. Approximation de Padé

Comme nous avons vu dans le premier chapitre, le modèle de Volterra deux noyaux peut s'écrire de la façon suivante :

$$\begin{aligned}\hat{Y}(t) = \tilde{H}_0\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t) + \frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_1\left(\left|\tilde{X}(t)\right|,\Omega\right).X(\Omega).e^{j\Omega t}.d\Omega \\ + \frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_2\left(\left|\tilde{X}(t)\right|,-\Omega\right).e^{j.2.\varphi_{\tilde{X}(t)}}.\tilde{X}^*(\Omega).e^{-j\Omega t}.d\Omega\end{aligned} \quad \text{(II-20)}$$

La première solution que nous avons utilisé pour arriver à des coûts de calcul comparables aux modèles classiques AM/AM et AM/PM est de synthétiser les noyaux dynamiques du modèle de Volterra deux noyaux sous la forme d'une approximation de Padé [II-20]. Cette approximation consiste à approcher une fonction donnée par son développement de Taylor dans un voisinage où elle est développable par des fractions rationnelles, dont le développement suivant les puissances croissantes de la variable coïncide avec celui de la fonction à l'ordre le plus élevé possible.

L'utilisation de cette technique nous permet d'éliminer le calcul de l'intégrale de convolution (procédure gourmande en ressources CPU) et d'exprimer la réponse du modèle en fonction des dérivées successives des signaux d'entrée et de sortie. Les équations constitutives

du modèle deviennent alors des équations différentielles qui doivent être résolues par l'une des nombreuses méthodes d'intégration numérique (Gear, Runge-Kutta, …).

Nous avons interpolé les noyaux de Volterra $H_1\left(\left|\tilde{X}(t)\right|,\Omega\right)$ et $H_2\left(\left|\tilde{X}(t)\right|,\Omega\right)$ sous la forme de fonctions rationnelles comme suit :

$$H\left(\left|\tilde{X}(t)\right|,\Omega\right)=\frac{\sum_{n=0}^{N}\alpha_n\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^n}{1+\sum_{m=1}^{M}\beta_m\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^m} \quad \text{avec} \quad N \le M \qquad \text{(II-21)}$$

Où N et M représentent respectivement l'ordre de l'approximation suivant la fréquence du numérateur et du dénominateur. Les coefficients $\alpha_n\left(\left|\tilde{X}(t)\right|\right)$ et $\beta_m\left(\left|\tilde{X}(t)\right|\right)$ de la décomposition en fonction rationnelle sont des fonctions statiques du signal d'entrée.

A partir de l'équation (II-21) on peut écrire :

$$\tilde{H}\left(\left|\tilde{X}(t)\right|,\Omega\right)=\sum_{n=0}^{N}\alpha_n\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^n-\tilde{H}\left(\left|\tilde{X}(t)\right|,\Omega\right).\sum_{m=1}^{M}\beta_m\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^m \quad \text{(II-22)}$$

En insérant l'équation (II-22) dans (II-20), on obtient alors:

$$\hat{Y}(t)=\tilde{H}_0\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t)+\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\left(\sum_{n=0}^{N_1}\alpha_{1n}\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^n-\tilde{H}_1\left(\left|\tilde{X}(t)\right|,\Omega\right).\sum_{m=1}^{M_1}\beta_{1m}\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^m\right).X(\Omega).e^{j\Omega t}.d\Omega$$
$$+\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\left(\sum_{n=0}^{N_2}\alpha_{2n}\left(\left|\tilde{X}(t)\right|\right).(-j.\Omega)^n-\tilde{H}_2\left(\left|\tilde{X}(t)\right|,-\Omega\right).\sum_{m=1}^{M_2}\beta_{2m}\left(\left|\tilde{X}(t)\right|\right).(-j.\Omega)^m\right).e^{j.2.\varphi_{\tilde{X}(t)}}.\tilde{X}^*(\Omega).e^{-j\Omega t}.d\Omega \qquad \text{(II-23)}$$

Pour faciliter l'écriture, on considère que :

$$\hat{Y}(t)=\tilde{H}_0\left(\left|\tilde{X}(t)\right|\right).\tilde{X}(t)+\tilde{Y}_{D1}(t)+\tilde{Y}_{D2}(t) \qquad \text{(II-24)}$$

où

$$\tilde{Y}_{D1}(t)=\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\sum_{n=0}^{N_1}\alpha_{1n}\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^n.X(\Omega).e^{j\Omega t}.d\Omega$$
$$-\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_1\left(\left|\tilde{X}(t)\right|,\Omega\right).\sum_{m=1}^{M_1}\beta_{1m}\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^m.X(\Omega).e^{j\Omega t}.d\Omega \qquad \text{(II-25)}$$

$$\tilde{Y}_{D2}(t)=\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\sum_{n=0}^{N_2}\alpha_{2n}\left(\left|\tilde{X}(t)\right|\right).(-j.\Omega)^n.e^{j.2.\varphi_{\tilde{X}(t)}}.\tilde{X}^*(\Omega).e^{-j\Omega t}.d\Omega$$
$$-\frac{1}{2\pi}\int_{-BW/2}^{+BW/2}\tilde{H}_2\left(\left|\tilde{X}(t)\right|,-\Omega\right).\sum_{m=1}^{M_2}\beta_{2m}\left(\left|\tilde{X}(t)\right|\right).(-j.\Omega)^m.e^{j.2.\varphi_{\tilde{X}(t)}}.\tilde{X}^*(\Omega).e^{-j\Omega t}.d\Omega \quad \text{(II-26)}$$

Sachant que, la somme d'intégrale est l'intégrale de la somme et en remarquant que :

$$\begin{cases}\tilde{Y}_{D1}(\Omega)=\tilde{H}_1\left(\left|\tilde{X}(t)\right|,\Omega\right).\tilde{X}(\Omega)\\ \tilde{Y}_{D2}(\Omega)=\tilde{H}_2\left(\left|\tilde{X}(t)\right|,-\Omega\right).\tilde{X}^*(\Omega)\end{cases} \quad \text{(II-27)}$$

on obtient :

$$\tilde{Y}_{D1}(t)=\frac{1}{2\pi}\sum_{n=0}^{N_1}\alpha_{1n}\left(\left|\tilde{X}(t)\right|\right)\int_{-BW/2}^{+BW/2}(j.\Omega)^n.X(\Omega).e^{j\Omega t}.d\Omega$$
$$-\frac{1}{2\pi}\sum_{m=1}^{M_1}\beta_{1m}\left(\left|\tilde{X}(t)\right|\right)\int_{-BW/2}^{+BW/2}\tilde{Y}_{D1}(\Omega).(j.\Omega)^m.e^{j\Omega t}.d\Omega \quad \text{(II-28)}$$

$$\tilde{Y}_{D2}(t)=\frac{1}{2\pi}\sum_{n=0}^{N_2}\alpha_{2n}\left(\left|\tilde{X}(t)\right|\right)\int_{-BW/2}^{+BW/2}(-j.\Omega)^n.\tilde{X}^*(\Omega).e^{j.2.\varphi_{\tilde{X}(t)}}e^{-j\Omega t}.d\Omega$$
$$-\frac{1}{2\pi}\sum_{m=1}^{M_2}\beta_{2m}\left(\left|\tilde{X}(t)\right|\right)\int_{-BW/2}^{+BW/2}\tilde{Y}_{D2}(\Omega).(-j.\Omega)^m.e^{j.2.\varphi_{\tilde{X}(t)}}.e^{-j\Omega t}.d\Omega \quad \text{(II-29)}$$

En considérant le fait que :

$TF^{-1}\left\{(j.\Omega)^k.F(\Omega)\right\}=\dfrac{d^k f(t)}{dt^k}$ où $f(t)$ est la transformée de Fourier inverse de $F(\Omega)$, et en prenant $N_1=N_2=N$ et $M_1=M_2=M$.

on obtient finalement :

$$\tilde{Y}_{D1}(t)=\sum_{n=0}^{N}\alpha_{1n}(\left|\tilde{X}(t)\right|).\frac{\partial^n\tilde{X}(t)}{\partial t^n}-\sum_{m=1}^{M}\beta_{1m}(\left|\tilde{X}(t)\right|).\frac{\partial^m\tilde{Y}_{D1}(t)}{\partial t^m} \quad \text{(II-30)}$$

$$\tilde{Y}_{D2}(t)=e^{j2\varphi_x(t)}\left(\sum_{n=0}^{N}\alpha_{2n}(\left|\tilde{X}(t)\right|).\frac{\partial^k\tilde{X}^*(t)}{\partial t^k}-\sum_{m=1}^{M}\beta_{2m}(\left|\tilde{X}(t)\right|).\frac{\partial^m\tilde{Y}_{D2}(t)}{\partial t^m}\right) \quad \text{(II-31)}$$

Le coût de calcul de ce modèle est alors 2(N+M)+1 fois celui du modèle classique AM/AM AM/PM. Un filtre d'ordre N+M = 5 est généralement suffisant, ce qui représente un coût de calcul du modèle de Volterra dynamique de l'ordre de 10 fois celui du modèle sans mémoire.

NB : Le calcul des coefficients $\alpha_n\left(\left|\tilde{X}(t)\right|\right)$ et $\beta_m\left(\left|\tilde{X}(t)\right|\right)$ est détaillé dans l'annexe 1.

Pour implémenter ce modèle dans le simulateur Scicos ou Simulink, nous devons résoudre les équations différentielles constituantes le modèle, en utilisant les techniques d'intégration numérique qui sont basées sur un calcul numérique des dérivées temporelles.

IV.2.1.1. Rappel sur la résolution numérique d'équations différentielles

Les méthodes d'intégration numérique permettent de transformer les équations différentielles en équations algébriques. Parmi celles utilisées, on distingue la méthode d'Euler rétrograde (*Backward Euler*) [II-21], de Runge-Kutta et celle de Gear. Toute ces méthodes procèdent à une discrétisation du temps, l'intervalle continu de simulation [0,T] est ainsi subdivisé en $n+k$ temps discrets t_i, où $i = n, n+1,.., n+k$ tels que:

$$t_{n+1} = t_n + h \tag{II-32}$$

Où h est le pas d'intégration, on note x_n la valeur approchée obtenue par l'application d'une méthode d'intégration.

Pour implémenter le modèle de Volterra deux noyaux, nous avons utilisé la méthode de Gear appelée aussi méthode BDF [II-22] (**Gear**,1971) (*Backward Differentiation Formula*s), cette méthode présente un excellent compromis entre la stabilité et la précision [II-23].

Supposons un signal $x(t)$ quelconque :

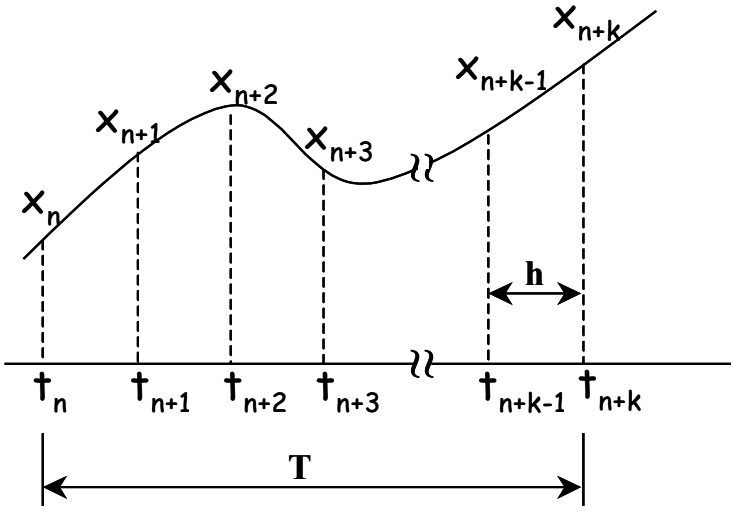


Figure II-14 : Signal discrétisé

La forme générale de la méthode BDF a la forme suivante :

$$x'_{n+k} = \frac{1}{h}\sum_{j=0}^{N} a_j x_{n+k-j} \tag{II-33}$$

où x'_{n+k} est la dérivée temporelle de x_{n+k}.

Des cas particuliers sont :

- k = 1 : $a_0 = -1$, $a_1 = 1$ (Méthode d'Euler) ;
- k = 2 : $a_0 = -\frac{3}{2}$, $a_1 = 2$, $a_2 = -\frac{1}{2}$;
- k = 3 : $a_0 = -\frac{11}{6}$, $a_1 = 3$, $a_2 = -\frac{3}{2}$, $a_3 = \frac{1}{3}$;
- k = 4 : $a_0 = -\frac{25}{12}$, $a_1 = 4$, $a_2 = -3$, $a_3 = \frac{4}{3}$, $a_4 = -\frac{1}{4}$.

En appliquant la formule de Gear d'ordre 2 dans (II-30) et (II-31), on obtient finalement la forme canonique suivante :

$$Y_{D1}(n) = \sum_{k=1}^{M1} c_{1k}(|X(n)|)X(n-k) - \sum_{k=1}^{N1} d_{1k}(|X(n)|)Y_{D1}(n-k) \tag{II-34}$$

$$Y_{D2}(n) = e^{j2\varphi_x(t)}\left(\sum_{k=1}^{M1} c_{2k}(|X(n)|)X(n-k) - \sum_{k=1}^{N1} d_{2k}(|X(n)|)Y_{D2}(n-k)\right) \tag{II-35}$$

Les figure (II-15) et (II-16) illustrent la représentation numérique des noyaux de Volterra dynamique.

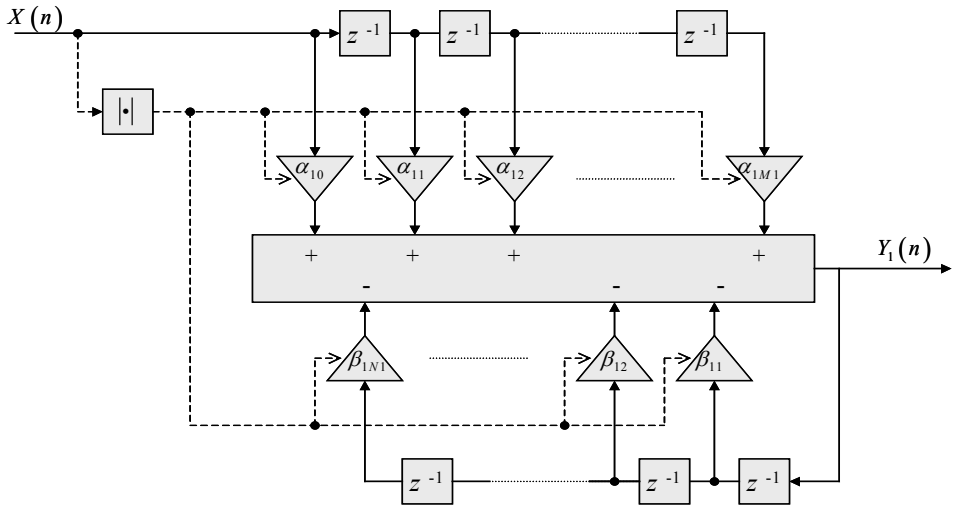


Figure II-15 : Diagramme du noyau de Volterra $Y_{D1}(n)$.

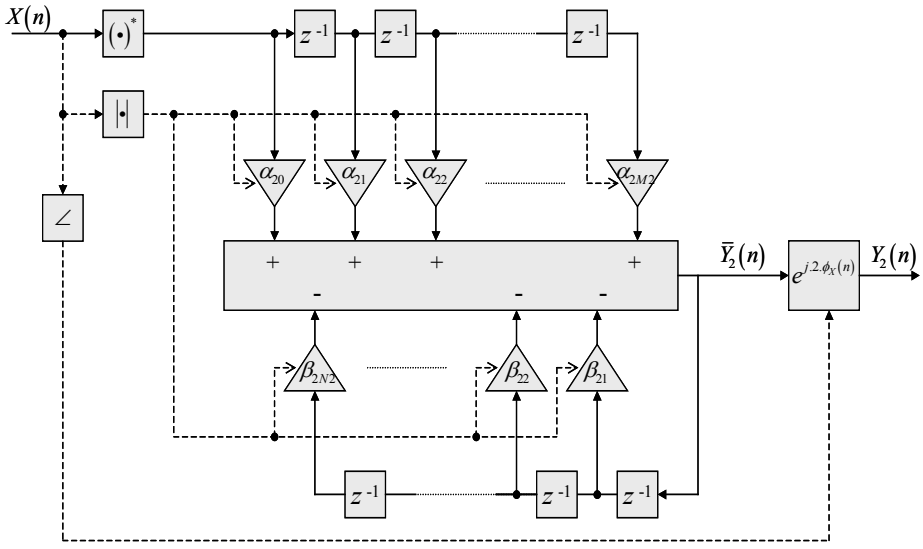


Figure II-16 : Diagramme du noyau de Volterra $Y_{D2}(n)$.

La méthode d'approximation de Padé peut poser des problèmes de stabilité numérique, notamment lorsque les ordres *N* et *M* utilisés au numérateur et au dénominateur sont élevés.

Pour garantir la stabilité de cette approximation, nous avons utilisé la technique d'identification des pôles-zéros reportée dans [II-24] pour chaque niveau de puissance d'entrée. Cette méthode consiste à analyser les pôles et zéros de la fonction approximée par l'approximation de Padé. Après nous localisons les pôles stables (partie réelle <0) de la fonction localisée, le modèle peut ensuite se mettre sous la forme d'une décomposition classique en éléments simples **[Annexe 2]**.

IV.3. IMPLEMENTATION DU MODELE DE VOLTERRA UN NOYAU (HF)

Considérons l'expression du modèle de Volterra un noyau :

$$\tilde{Y}(t) = \frac{1}{2\pi}.\int_{-BW/2}^{BW/2} H_V\left(\left|\tilde{X}(t)\right|,\Omega\right).\tilde{X}(\Omega).e^{j\Omega t}.d\Omega \qquad \text{(II-36)}$$

La deuxième solution que nous avons adopté afin d'éliminer l'intégrale de Fourier consiste à décomposer le noyau de Volterra en série de fonctions séparables comme ci-dessous :

$$H_V\left\{\left|\tilde{X}(t)\right|,\Omega\right\} = \sum_{k=0}^{K} \alpha_k(\Omega).f_k\left(\left|\tilde{X}(t)\right|\right) \qquad \text{(II-37)}$$

où $f_k(\cdots)$ est une fonction de base arbitraire, notamment un polynôme d'ordre k.

Cette décomposition permet de dissocier les deux variables de la fonction, une fonction dépend de la fréquence, l'autre du signal d'entrée. On peut alors considérer la décomposition précédente et réécrire l'expression du modèle de la manière suivante :

$$\tilde{Y}(t) = \frac{1}{2.\pi} \sum_{k=0}^{K} .f_k \left\{ \left| \tilde{X}(t) \right| \right\} . \int_{-BW/2}^{+BW/2} \alpha_k(\Omega) . \tilde{X}(\Omega) . e^{j.\Omega t} . d\tau \qquad \text{(II-38)}$$

La figure suivante représente symboliquement la nouvelle architecture du modèle de Volterra un noyau. La structure du modèle correspond alors à la mise en parallèle des cellules élémentaires dites de Wiener (un filtre linéaire $\alpha_k(\cdots)$ suivi par une non linéarité statique $f_k(\cdots)$).

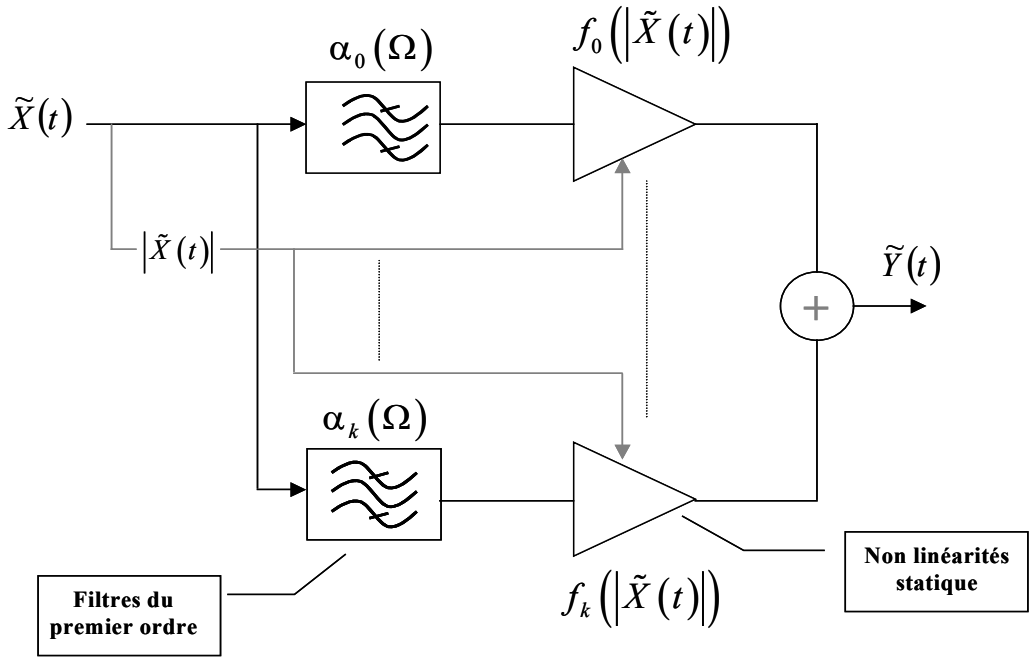


Figure II-17 : Structure du modèle de Volterra à un noyau

IV.4. MODELE A REPONSE IMPULSIONNELLE NON LINEAIRE

Considérons l'expression du modèle à réponse impulsionnelle non linéaire statique :

$$\tilde{Y}(t) = \int_{0}^{Tm} h_R \left\{ \left| \tilde{X}(t-\tau) \right|, \tau \right\} . \tilde{X}(t-\tau) . d\tau \qquad \text{(II-39)}$$

En utilisant la décomposition présentée dans le paragraphe précédant, la réponse impulsionnelle $h_R(\cdots)$ s'écrit :

$$h_R\left\{\left|\tilde{X}\left(t-\tau\right)\right|,\tau\right\}=\sum_{l=0}^{L}\varphi_l\left(\tau\right).f_l\left(\left|\tilde{X}\left(t-\tau\right)\right|\right) \qquad \text{(II-40)}$$

On peut alors reformuler l'expression du modèle comme suit :

$$\tilde{Y}\left(t\right)=\sum_{k=0}^{K}\int_{0}^{Tm}\varphi_k\left(\tau\right).f_k\left\{\left|\tilde{X}\left(t-\tau\right)\right|\right\}.d\tau \qquad \text{(II-41)}$$

où $f_k\left(\left|\tilde{X}\left(t-\tau\right)\right|\right)=f_l\left(\left|\tilde{X}\left(t-\tau\right)\right|\right).\tilde{X}\left(t-\tau\right)$.

Cette expression est la somme de produits de convolution linéaire entre les fonctions $\varphi_k\left(t\right)$ et des fonctions dépendant du signal d'entrée $f_k\left(\cdots\right)$. On peut réécrire cette expression dans le domaine fréquentiel comme ci-dessous.

$$\tilde{Y}\left(t\right)=\sum_{k=0}^{K}TF^{-1}\left\{\tilde{\alpha}_k\left(\Omega\right)\ .\ \tilde{F}_k\left(\Omega\right)\right\}=\sum_{k=0}^{K}\tilde{Y}_k\left(t\right) \qquad \text{(II-42)}$$

où $\tilde{\alpha}_k\left(\ \right)$ correspond à la transformée de Fourier du coefficient d'ordre *k* de la décomposition de la réponse impulsionnelle non linéaire et $\tilde{F}_k\left(\ \right)$ à la transformée de Fourier de la fonction de base.

La structure du modèle revient à mettre en parallèle des cellules élémentaires de Hammerstein (une non linéarité statique $f_k\left(\cdots\right)$ suivi d'un filtre linéaire $\varphi_k\left(\cdots\right)$) mises en parallèle, comme l'illustre la figure II-18. Ici, à la différence du modèle de Volterra un noyau, les non linéarités statiques sont placées en amont des filtres du premier ordre.

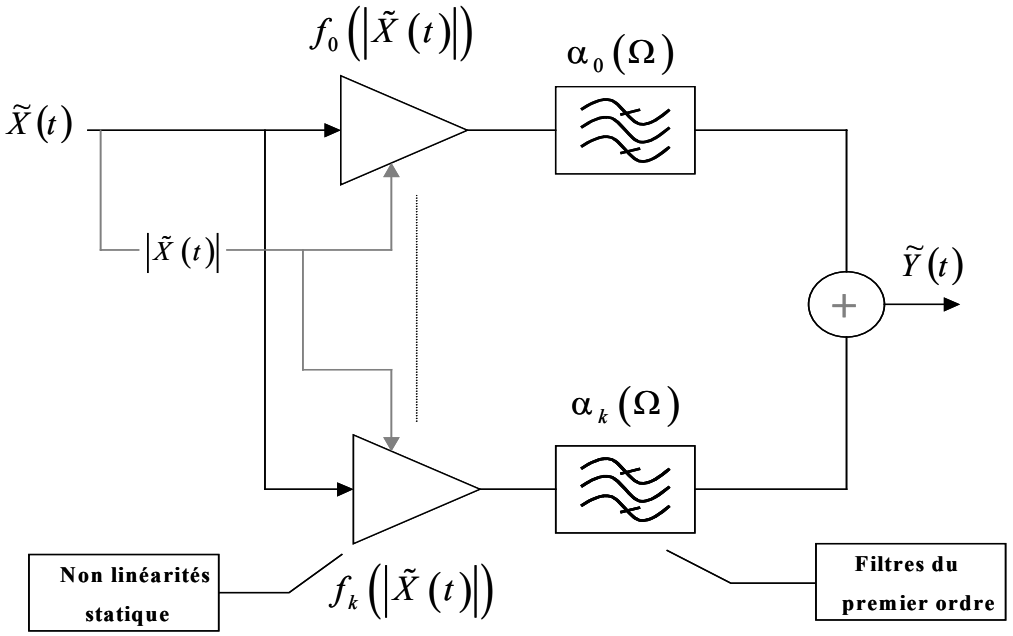


Figure II-18 : Structure du Modèle à réponse impulsionnelle non linéaire

IV.5. SYNTHESE DES FILTRES LINEAIRES

Pour synthétiser les filtres linéaires présents dans ces modèles, nous avons expérimenté deux méthodes. La première méthode consiste à synthétiser les filtres linéaires sous la forme d'une fonction rationnelle (approximation de Padé) vue dans le paragraphe IV.2.1.

$$\alpha(\Omega) = \frac{\sum_{k=0}^{M} \varphi_k . (j.\Omega)^k}{1 + \sum_{k=1}^{N} \beta_k . (j.\Omega)^k} \qquad \text{(II-43)}$$

Cette approximation permet un passage trivial entre les domaines fréquentiel et temporel et l'obtention de d'équations différentielles. Après on utilise la méthode de *Gear* pour approximer les dérivées temporelles des signaux, on aboutit au schéma classique de filtre numérique présenté sur la figure ci-dessous.

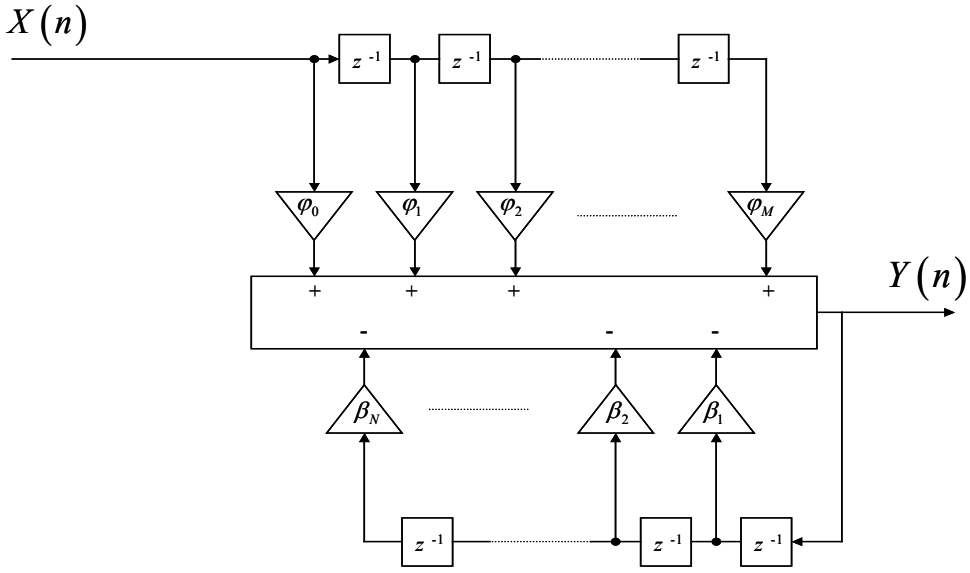


Figure II-19 : Structure du filtre linéaire

Comme nous avons mentionné précédemment, l'approximation de Padé peut présenter de temps à autre des petits problèmes de stabilité numérique.

La deuxième méthode que nous avons utilisé pour synthétiser les filtres consiste à décomposer les filtres en pôles/résidus. Cette méthode présente une excellente stabilité numérique car aucune dérivée temporelle d'ordre élevé n'a besoin d'être calculée.

$$\alpha(\Omega) = \mu \ . \ \frac{\prod_{n=1}^{N} (j.\Omega - z_n)}{\prod_{m=1}^{M} (j.\Omega - p_m)} \qquad \text{(II-44)}$$

Si $N < M$ nous pouvons décomposer explicitement cette équation en éléments simples comme ci-dessous :

$$\alpha(\Omega) = \frac{A_1}{j.\Omega - p_1} + \frac{A_2}{j.\Omega - p_2} + \cdots + \frac{A_M}{j.\Omega - p_M} \qquad \text{(II-45)}$$

On aboutit ainsi à une structure de filtres de premier ordre en parallèle. On peut ensuite efficacement utiliser une méthode d'intégration récursive comme la méthode de Backward Euler.

IV.6. PROCEDURE D'INTEGRATION DU MODELE

Comme nous avons mentionné dans le paragraphe II-4-3 et III-2-3, l'implémentation des nouveaux modèles dans l'environnement Simulink se fait à l'aide des blocs S-Functions (system-functions) et dans l'environnement Scicos à l'aide des blocs Generic.

La figure II-20 montre le principe de l'implémentation de ces modèles comportementaux dans l'environnement Simulink et Scicos, partant de la procédure de caractérisation (mesure ou simulation) jusqu'à la mise en œuvre complète du modèle.

Les fichiers de départ sont issus de mesure ou de simulation, et doivent avoir un format standard de type (.dat). Ces données d'extraction du modèle contiennent toutes les informations nécessaires à la modélisation. Après quelques traitements numériques, le modèle est intégralement défini par un fichier header (.head) contenant les informations sur les coefficients de l'approximation de Padé dans le cas du modèle de Volterra deux noyaux et celles des filtres linéaires dans le cas du modèle de Volterra un noyau et du modèle à réponse impulsionnelle non linéaire.

Au cours de la simulation, les algorithmes qui représentent les modèles à mémoire lisent ces fichiers header. Ces algorithmes implémentés dans Simulink et Scicos utilisent une interpolation par les Splines cubiques développées à IRCOM.

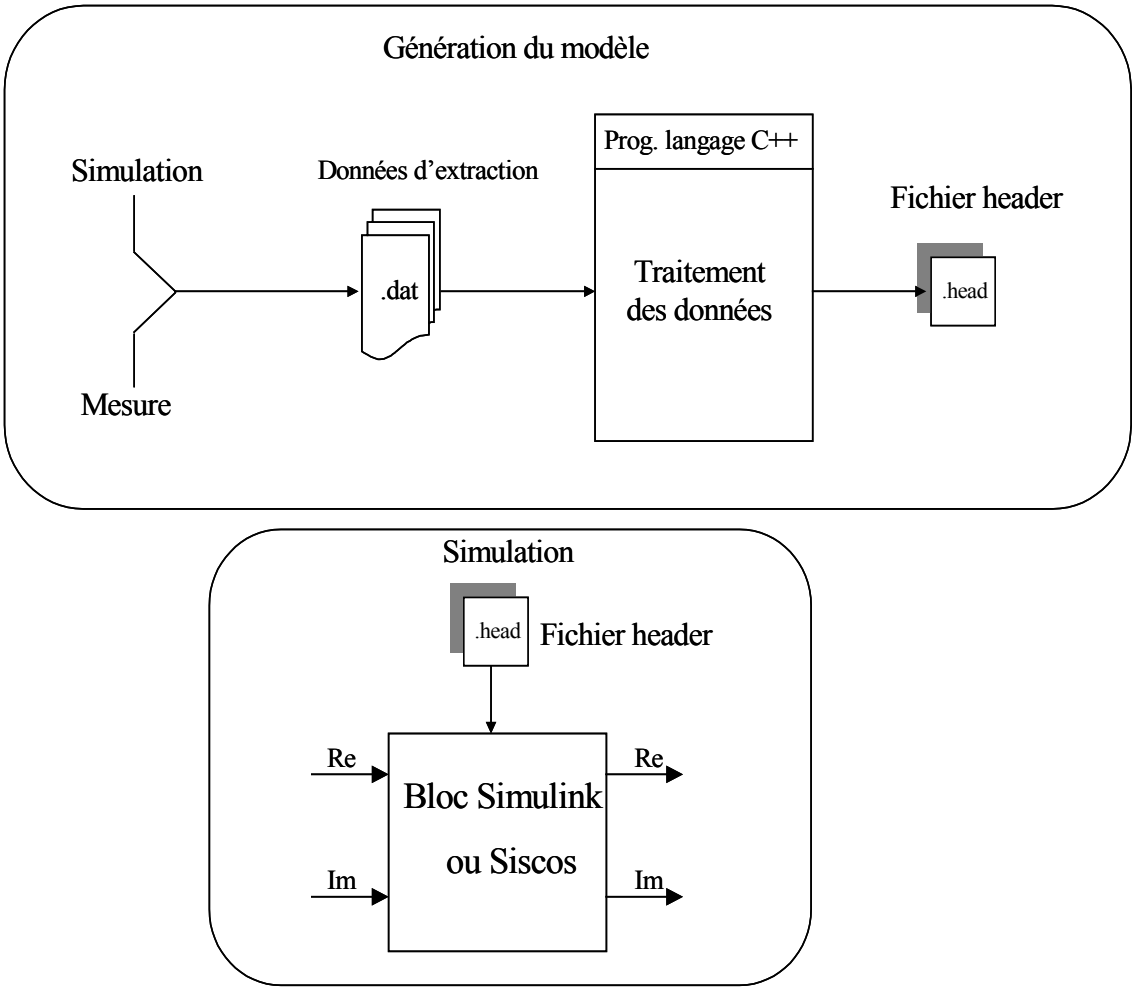


Figure II-20 : Synoptique de génération du modèle et de la simulation

La figure ci-dessous montre le format du fichier head du modèle de Volterra un noyau et le modèle à réponse impulsionnelle non linéaire. Le fichier comprend deux parties, la première commence par l'entête où nous définissons le type du modèle (modèle de Volterra un noyau ou à réponse impulsionnelle non linéaire), les valeurs des impédances d'entrée et de sortie, l'ordre de développement de la fonction de base, la bande d'extraction du modèle et la fréquence centrale. La deuxième partie contient toutes les informations sur les filtres linéaires (pôles et résidus) [II-25].

```
//
S_MODEL AMPLI_LNA_BIL
{
   EQUIPMENT_TYPE  = AMPLIFIER,
   MODEL_TYPE      = VOLTERRA,
   Z1_REF          = [5.000000E+001, 0.000000E+000],
   Z2_REF          = [5.000000E+001, 0.000000E+000],
   MISMATCH_IGNORE = 0,
   DC_FLAG         = 0,
   BASIS_ORDER     = 10,
   Y11_ORDER       = [7,7,8,8,8,8,8,8,8,8],
   Y12_ORDER       = [8,8,8,8,8,8,8,9,9,9],
   Y12_P_ORDER     = [8,8,6,8,8,10,10,10,10,10],
   Y21_ORDER       = [4,6,6,8,6,6,5,5,5,5],
   Y22_ORDER       = [8,8,10,10,7,7,7,7,7,7],
   Y22_P_ORDER     = [8,10,10,10,10,8,8,8,8,8],
   DC1_ORDER       = [10,10,10,10,10,10,10,10,10,10],
   DC2_ORDER       = [10,10,10,10,10,10,10,10,10,10],
   A1_RANGE        = [1.414214E-003, 4.101219E-002],
   BANDWIDTH       = [1.860000E+009, 2.060000E+009],
   CARRIER1        = 1.960000E+009,
   CARRIER2        = 1.960000E+009,
   FREQ_NORM       = 6.283185E+008,

   RES11           = [-1.5057031626E+001 , -9.0407033359E+000 , -1.3247831636E+000 , +8.3694843087E+000 , +2.7661040249E-001
                      +7.8879055219E+001 , +3.3499320154E+001 , +2.9782075964E+001 , -6.1546207719E+001 , -3.1025969359E-001
                      -4.0599145255E+002 , -1.0918900300E+002 , -2.3610214416E+002 , +2.9088226692E+002 , -1.3517512097E-003
                      +1.3459187482E+003 , +4.0052516906E+002 , +9.1477662089E+002 , -1.0018754066E+003 , -6.0520460925E-001
                      -2.7562575481E+003 , -7.7846082295E+002 , -2.2384654159E+003 , +2.0329525826E+003 , +1.3241892419E-001
                      +3.4099614129E+003 , +7.4586932279E+002 , +3.7821461349E+003 , -2.3827504802E+003 , +9.9428216388E-001
                      -2.2334084113E+003 , -1.6586155951E+002 , -4.4787258584E+003 , +1.4623242609E+003 , -4.2158610658E+000
                      +3.6130465188E+002 , -3.8376054836E+002 , +3.6115891802E+003 , -2.4519147708E+002 , +5.7706446431E+000
                      +3.9960398161E+002 , +3.8836506367E+002 , -1.7775022483E+003 , -1.7210128614E+002 , -3.7208035206E+000
                      -1.8135311341E+002 , -1.2277585521E+002 , +4.0100389320E+002 , +6.4878112347E+001 , +9.5436497256E-001
   POLES11         = [-2.4239125421E+000 , +2.8724867687E+000 , -2.4239125421E+000 , -2.8724867687E+000 , -4.5990798357E-001
                      -1.6435896272E+000 , +2.8710907818E+000 , -1.6435896272E+000 , -2.8710907818E+000 , -3.9458449748E-001
                      -1.5461102676E+000 , +3.1892128545E+000 , -1.5461102676E+000 , -3.1892128545E+000 , -1.7648356520E-001
                      -1.5441920305E+000 , +3.1920269711E+000 , -1.5441920305E+000 , -3.1920269711E+000 , -3.5114784617E-001
                      -1.5420562422E+000 , +3.2042278466E+000 , -1.5420562422E+000 , -3.2042278466E+000 , -2.5126713436E-001
                      -1.5527947010E+000 , +3.2336560339E+000 , -1.5527947010E+000 , -3.2336560339E+000 , -3.0078753247E-001
                      -1.5607837560E+000 , +3.2576800612E+000 , -1.5607837560E+000 , -3.2576800612E+000 , -3.0135131039E-001
                      -1.5705097168E+000 , +3.2840933652E+000 , -1.5705097168E+000 , -3.2840933652E+000 , -2.9745409831E-001
                      -1.5838336421E+000 , +3.3156470832E+000 , -1.5838336421E+000 , -3.3156470832E+000 , -2.9445191761E-001
                      -1.6042065225E+000 , +3.3567088781E+000 , -1.6042065225E+000 , -3.3567088781E+000 , -2.9287324223E-001
   RES12           = [-4.4918624595E-001 , +3.5387903135E+000 , -1.1616168402E+000 , +1.0462984576E+000 , -4.1747121390E-003
                      +1.0430857019E+001 , -1.1152859363E+001 , +7.2013096988E+000 , -1.2026499619E+001 , -1.9950113983E-004
                      -1.0336381416E+002 , +3.9536066003E+001 , -4.2183029556E+001 , +9.5556190328E+001 , -1.5910518090E-002
                      +5.2298896162E+002 , -1.3197138765E+002 , +1.7682604674E+002 , -4.5205159993E+002 , -3.0401303073E-001
                      -1.5529270916E+003 , +3.4785096518E+002 , -4.8567056484E+002 , +1.3144554233E+003 , +1.8017768422E+000
                      +2.8859505332E+003 , -6.3577100546E+002 , +8.6921026356E+002 , -2.4382856051E+003 , -3.3391919110E+000
                      -3.4098586603E+003 , +7.5264734193E+002 , -1.0112117371E+003 , +2.9041739578E+003 , +2.3187829556E+000
                      +2.4934591594E+003 , -5.5070042588E+002 , +7.4369051704E+002 , -2.1502514336E+003 , +7.4381990373E-002
                      -1.0290669483E+003 , +2.2195136050E+002 , -3.1114640180E+002 , +9.0185385346E+002 , -8.2070156261E-001
```

Figure II-21 : Format du fichier head

V. Validation des modèles

V.1. MODELE DE VOLTERRA DEUX NOYAUX

V.1.1. Application au cas d'un HPA en bande K

Nous avons testé le modèle de Volterra deux noyaux sur un amplificateur trois étages fonctionnant en bande K (19 GHz). Cet amplificateur est conçu autour de transistors de type pHEMT de 8*75*0.25 μm de développement de grille. Sa puissance de sortie nominale est de 1 Watt. Nous présentons sur la figure suivante sa topologie générale.

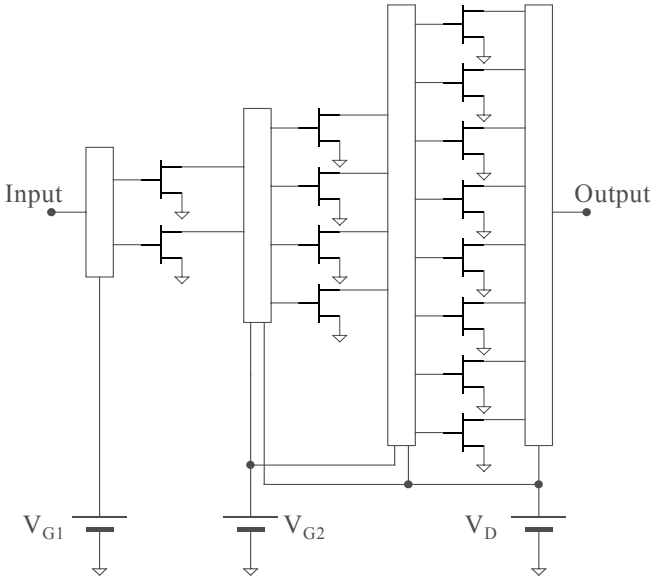


Figure II-22 : Topologie de l'amplificateur bande K (19 GHz)

V.1.1.1. Extraction du modèle

Le modèle de cet amplificateur a été extrait à partir de son schéma de conception avec le simulateur circuit Xepedion/Goldengate™ dont le noyau de simulation (LISA) a été développé au laboratoire de l'IRCOM à Limoges.

Pour cet amplificateur, les figures II-23 et II-24 montrent les courbes de gain et de déphasage entrée/sortie en fonction de la puissance et de la fréquence obtenues à l'aide de la méthode de l'équilibrage harmonique.

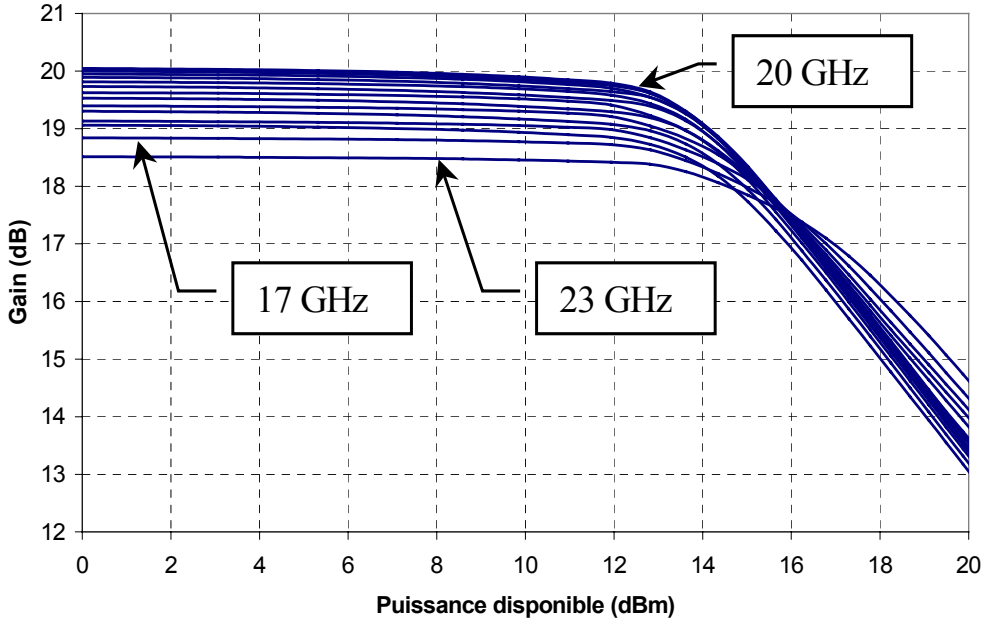


Figure II-23 : Gain de l'amplificateur en fonction de la puissance d'entrée

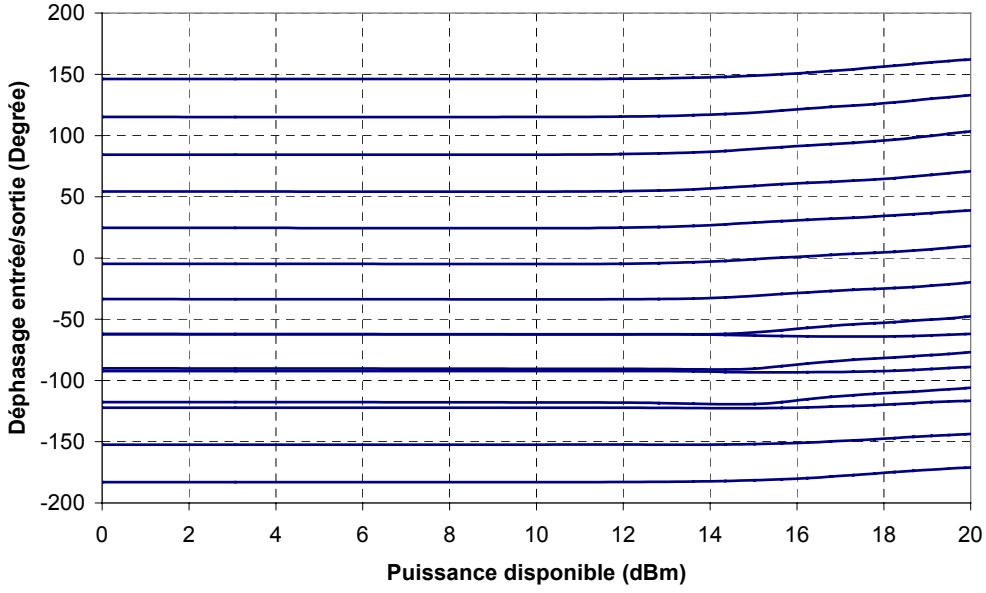


Figure II-24 : Déphasage entrée/sortie en fonction de la puissance d'entrée

Les noyaux dynamiques du modèle ont été extraits également à partir du simulateur Xepedion/Goldengate™ en équilibrage harmonique sur une bande de 200 MHz autour de la fréquence de référence F_0.

Les figures suivantes illustrent l'allure des noyaux dynamiques (en partie réelle et imaginaire) en fonction de la puissance d'entrée paramétrée par l'écart de fréquence et l'approximation de Padé avec N= 2 et M=3.

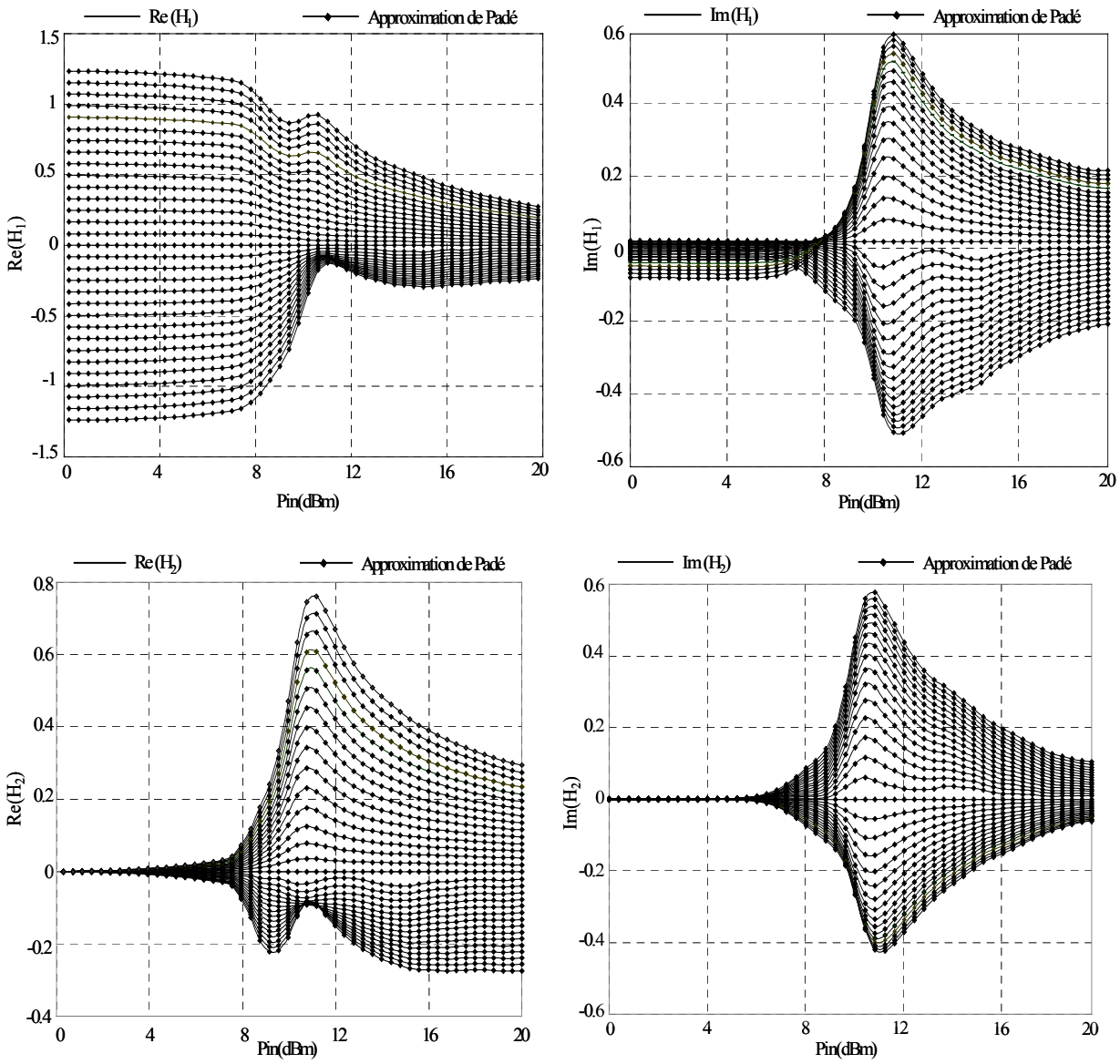


Figure II-25 : H1 et H2 en partie Réelle/Imaginaire pour différentes fréquences

La figure II-25 montre aussi la qualité de l'approximation de Padé. On peut remarquer un bon accord entre les noyaux dynamiques du modèle et cette approximation pour N = 2 et M = 3 [II-26].

V.1.1.2. Validation du modèle

Afin de valider l'implémentation du modèle de Volterra deux noyaux dans Scicos et Simulink, nous avons essayé de reproduire les caractéristiques de ce modèle.

Pour cela une série de simulations biporteuse est effectuée (un signal d'entrée comportant deux fréquences $x(t) = X_0.e^{j\omega_0 t} + \delta X.e^{j(\omega_0+\Omega)t}$ *où* $\delta X \ll X_0$). Le résultat de ces simulations est

comparé à la réponse du modèle comportemental excité pour les mêmes points caractéristiques. Dans les deux figures suivantes les résultats de ces comparaisons ont été tracés.

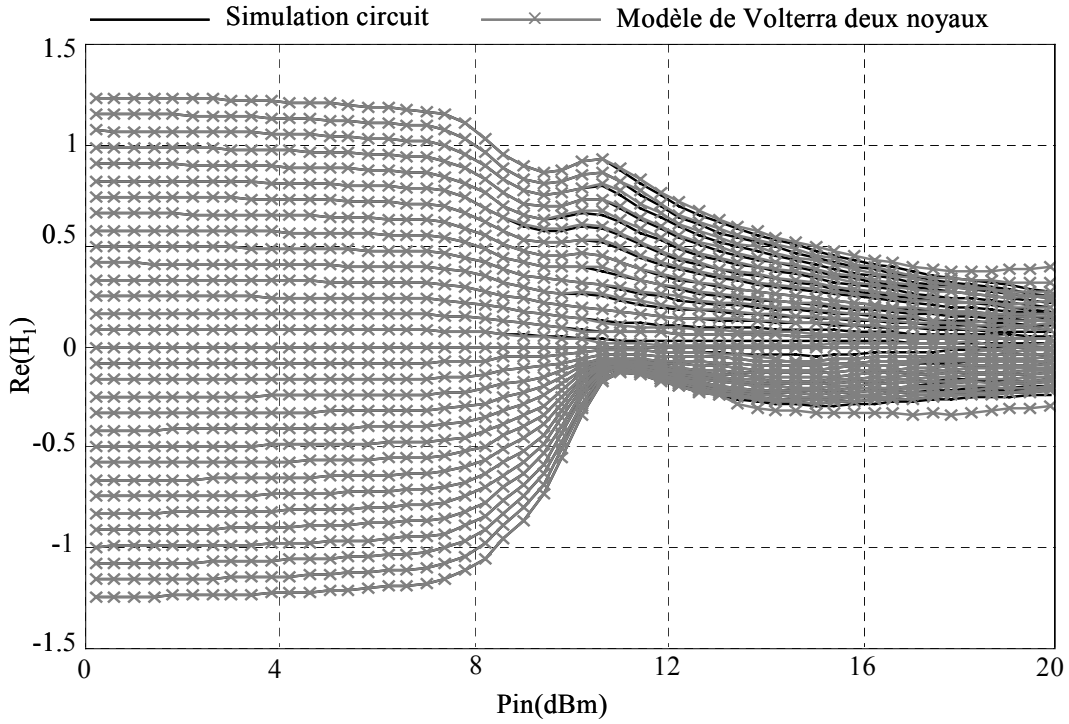


Figure II-26 : Partie réelle de H_1 pour différentes fréquences

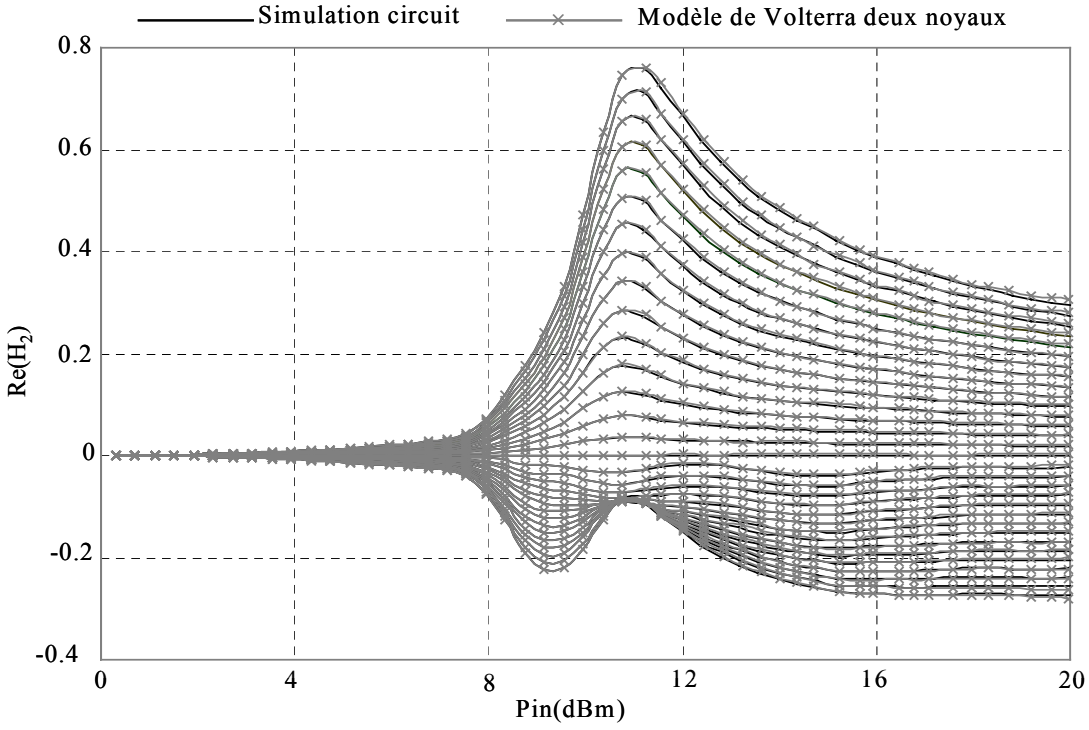


Figure II-27 : Partie réelle de H_2 pour différentes fréquences

Les résultats donnés dans les deux figures précédentes permettent de se rendre compte que le modèle de Volterra deux noyaux reproduit les variations des deux noyaux dynamiques de l'amplificateur sur l'ensemble de la bande de l'amplificateur (+/- 100 MHz autour de 19 GHz GHz) jusqu'à des niveaux de puissance d'entrée correspondant à 11 dBm d'IBO (zone non linéaire).

V.1.1.3. Intégration du modèle dans une chaîne de communication sous Scicos

Nous avons testé ce modèle dans une chaîne de communication (Figure II-28) développée sous Scicos par A. Layec [II-27], il s'agit d'un émetteur récepteur avec un canal bruité.

La simulation de cette chaîne de communication est réalisée en bande de base. Le signal d'information de cette chaîne est un signal QPSK. Le premier bloc est le générateur binaire pseudo-aléatoire, ce générateur fournit un vecteur d'éléments qui représente les valeurs de symboles transmises. Le bloc sur-échantillonneur et Convolution sont utilisés pour réaliser un filtre à réponse impulsionnelle finie, avec la réponse impulsionnelle du RRC (Root Raised Cosine) sur les deux canaux. L'amplification du signal est effectuée par l'amplificateur de puissance. Le vecteur de bruit gaussien blanc est ajouté au signal de sortie de l'amplificateur. Afin de respecter le critère de Nyquist un second filtre RRC est exécuté sur les composants reçus, et un filtrage adaptatif est réalisé pour réduire au minimum les effets du bruit. Le démodulateur PSK fournit les vecteurs symbole démodulés.

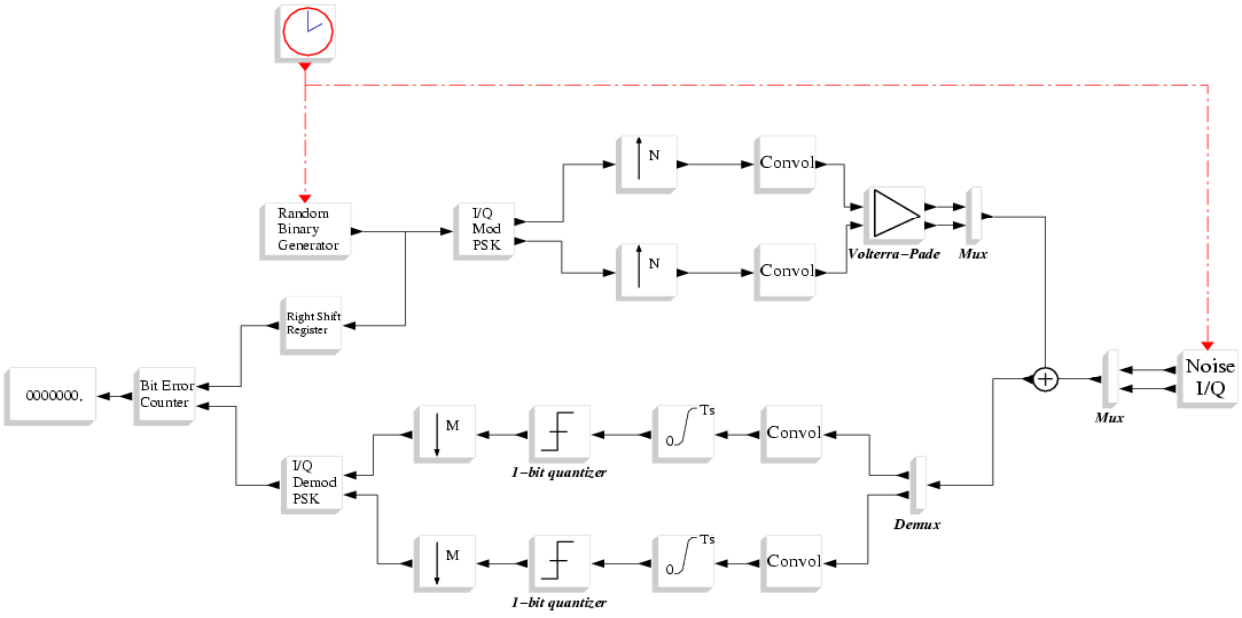


Figure II-28 : Diagramme de la chaîne de communication dans l'environnement Scicos

Comme exemple d'application nous avons calculé le taux d'erreur binaire (TEB) en comptant le nombre de bits erronés reçus et en effectuant le rapport avec le nombre de bits total transmis. Le temps de simulation pour une évaluation complète de TEB de la chaîne de transmission est d'environ 40 heures en utilisant le modèle de Volterra deux noyaux et 4 heures en utilisant le modèle classique AM/AM et AM/PM.

La figure II-29 représente le taux d'erreur binaire en fonction du rapport signal sur bruit (Eb/No) obtenu à l'aide du modèle de Volterra deux noyaux et le modèle classique AM/AM et AM/PM.

Nous remarquons sur la figure suivante la dégradation du TEB due au modèle classique AM/AM et le modèle de Volterra deux noyaux, nous remarquons également que l'utilisation d'un modèle d'amplificateur qui prend en considération les effets de mémoire, montre un TEB plus dégradé.

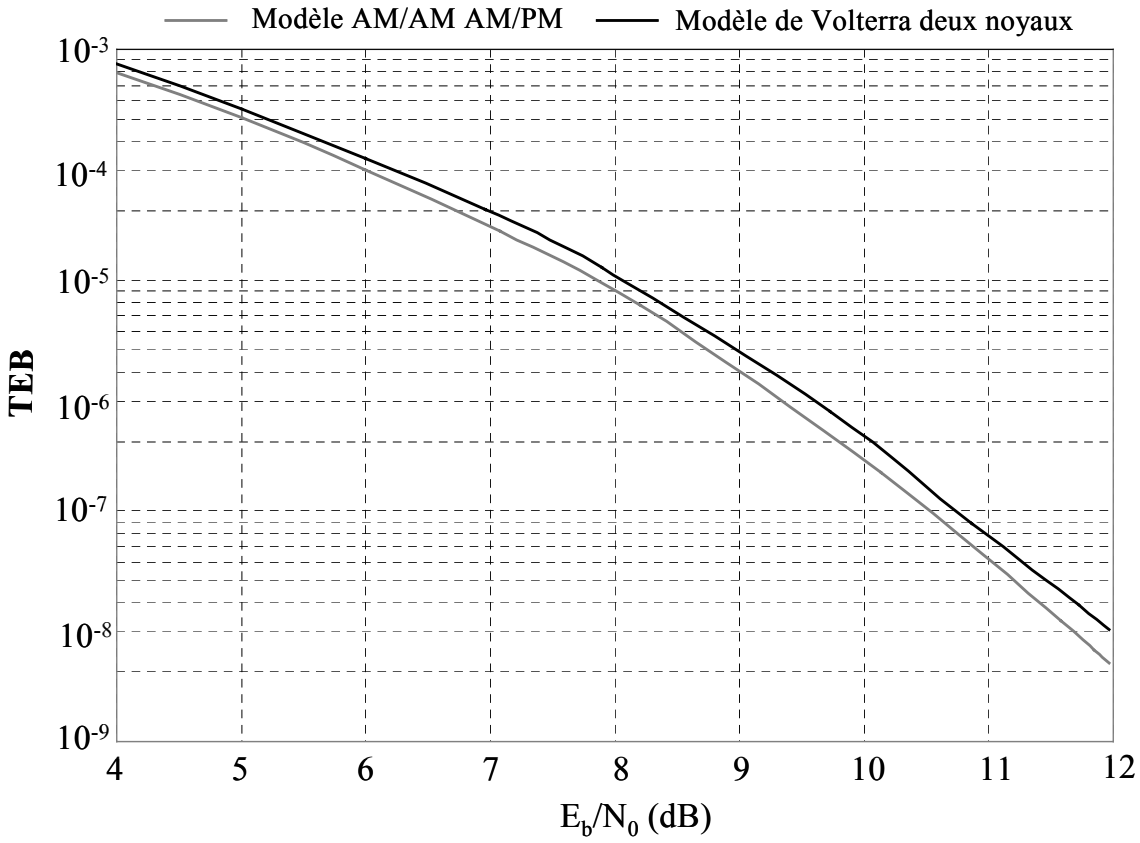


Figure II-29 : Taux d'erreur binaire en fonction du rapport signal sur bruit

V.2. MODELE DE VOLTERRA UN NOYAU

V.2.1. Application au cas d'un amplificateur à faible bruit (LNA)

Le modèle de Volterra un noyau et le modèle à réponse impulsionnelle non linéaire ont été testé sur un amplificateur LNA de technologie BiCMOS fonctionnant en bande L (autour de 1.96 GHz), possédant une boucle de contrôle du gain.

V.2.1.1. Caractéristique de l'amplificateur

Nous présentons sur les figures suivantes le gain et le déphasage entrée/sortie de l'amplificateur en fonction de la puissance d'entrée.

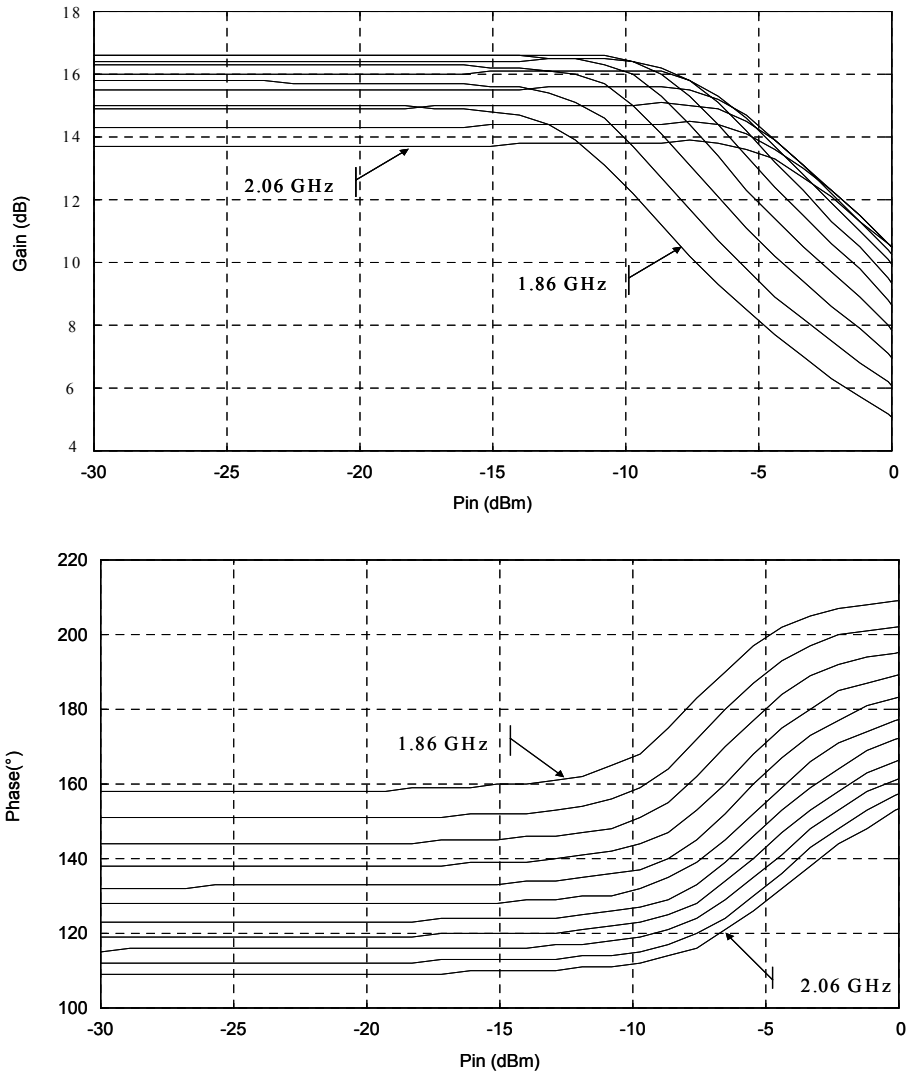


Figure II-30 : Gain et déphasage entrée/sortie de l'amplificateur en fonction de la puissance d'entrée

Comme le montre les figures suivantes, les courbes sont extraites sur une dynamique de puissance de 30 dBm.

V.2.1.2. Extraction du modèle de Volterra un noyau

Le noyau dynamique de l'amplificateur a été extrait à l'aide d'une série de courbes AM/AM et AM/PM sur une bande de +/- 100 MHz autour de 1.96 GHz.

Les figures suivantes illustrent dans cette bande les variations du noyau dynamique de l'amplificateur en fonction de la puissance d'entrée puis de l'écart en fréquence.

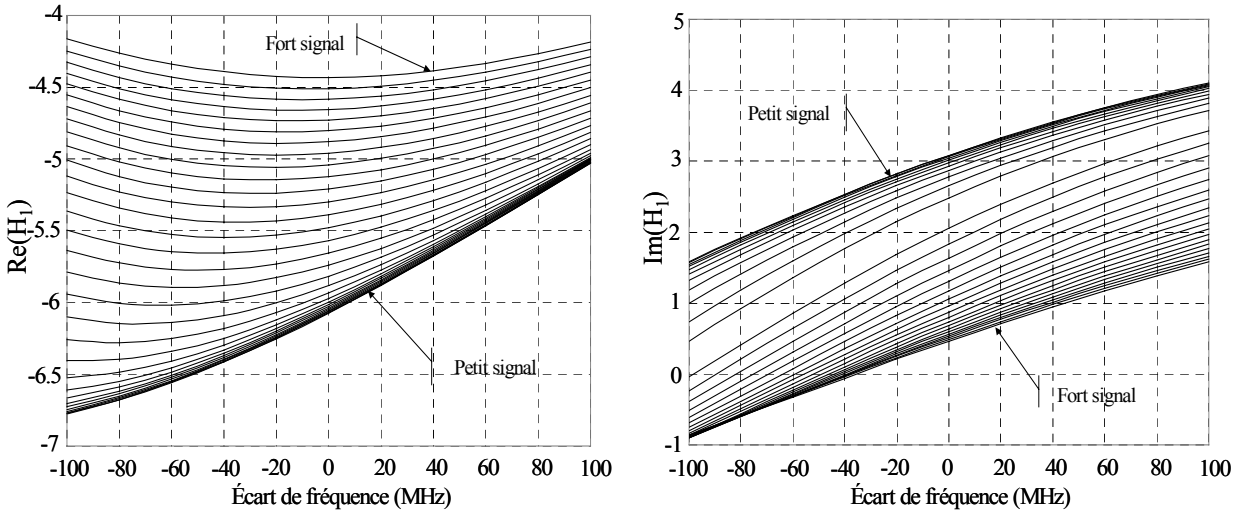


Figure II-31 : Partie réelle/imaginaire de H1 pour différentes puissances d'entrée

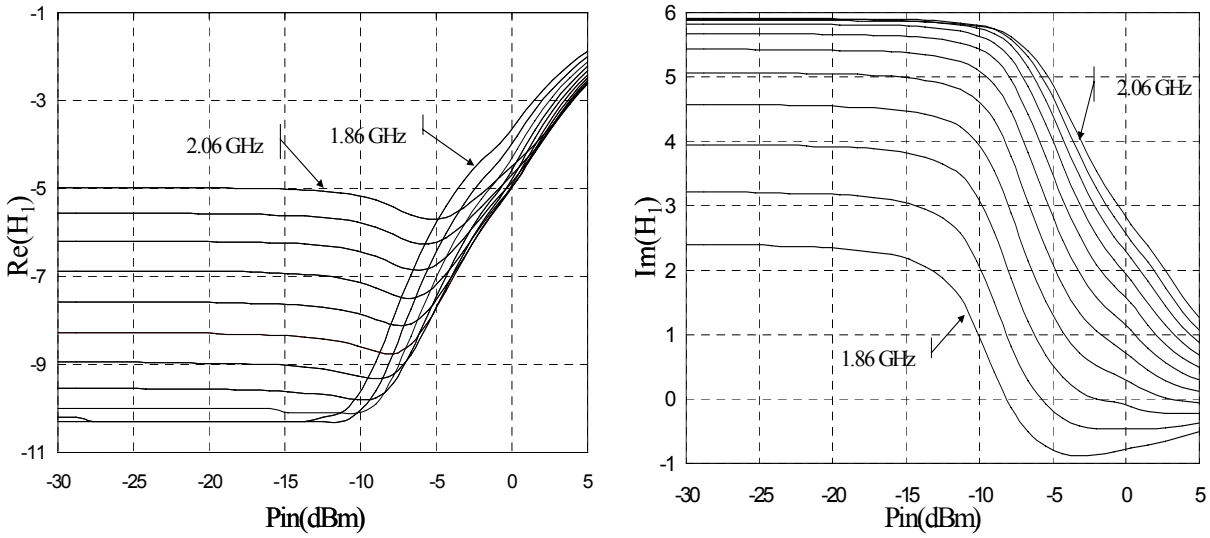


Figure II-32 : Partie réelle/imaginaire de H1 pour différentes fréquences d'entrée

V.2.1.3. Validation du modèle

Pour valider l'implémentation du modèle de Volterra un noyau, nous avons donc comparé la réponse en monoporteuse du circuit avec celle du modèle de Volterra un noyau pour l'ensemble de la bande utile (figure II-33 à figure II-36). Pour cela une série de simulations monoporteuse est effectuée.

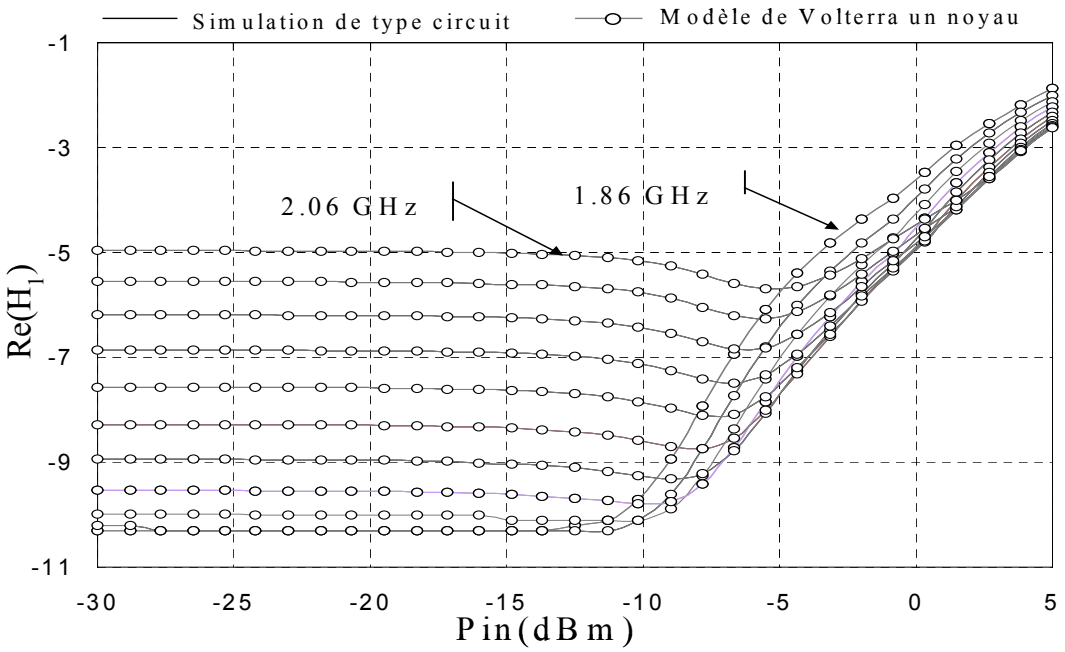


Figure II-33 : Partie réelle de H1 pour différentes fréquences

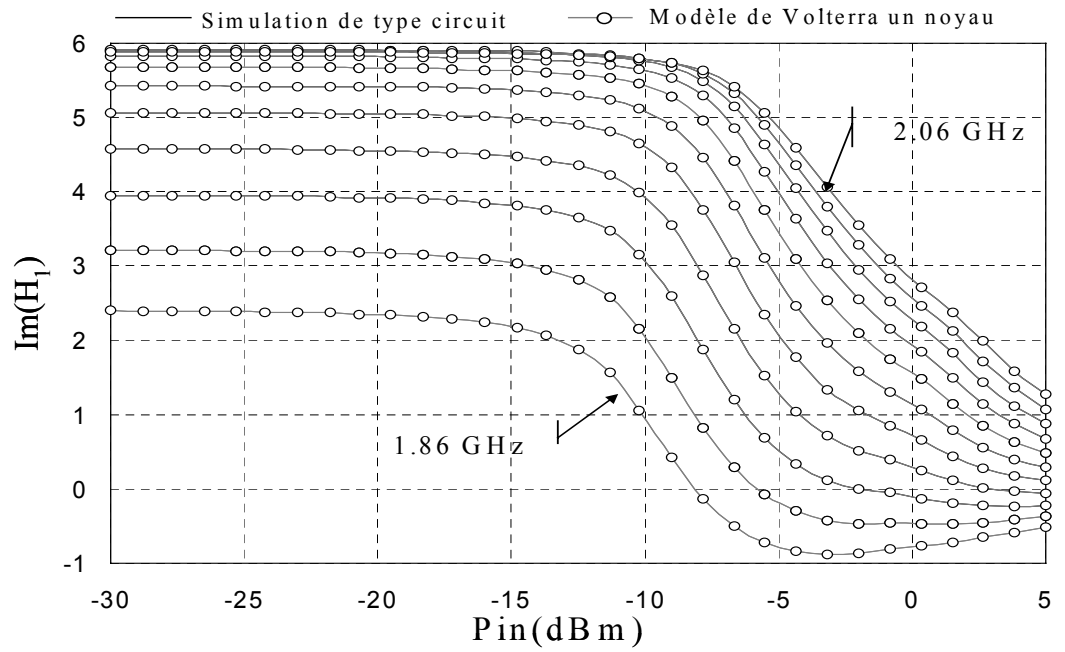


Figure II-34 : Partie imaginaire de H1 pour différentes fréquences

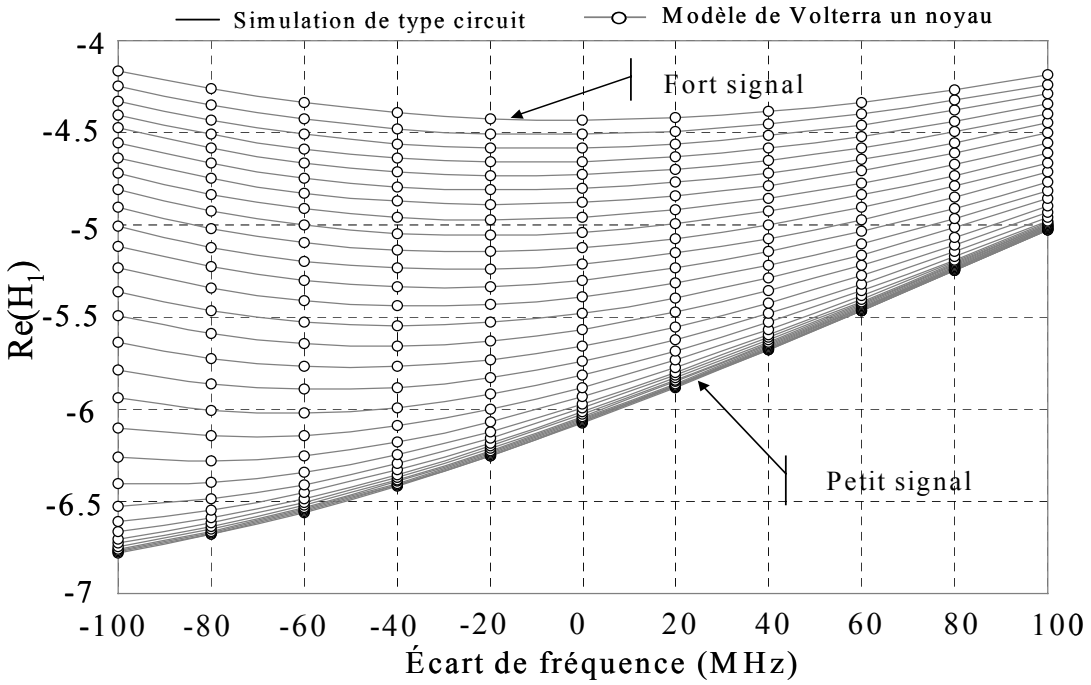


Figure II-35 : Partie réelle de H1 pour différentes puissances d'entrée

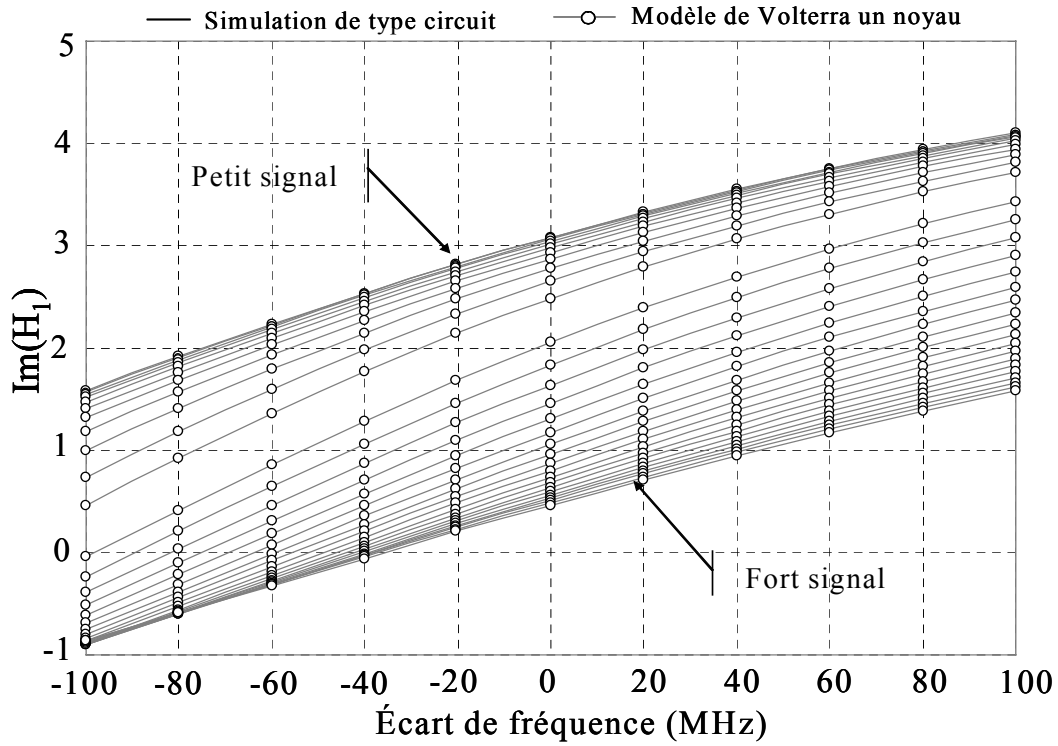


Figure II-36 : Partie imaginaire de H1 pour différentes puissances d'entrée

Nous remarquons que le modèle à un noyau a une très bonne concordance avec le circuit de l'amplificateur, cela permet d'assurer que la mémoire haute fréquence a bien été intégrée dans le modèle. Ce modèle a été extrait directement à partir des données monoporteuses, il est logique que ces caractéristiques soient bien présentées reproduites.

V.3. MODELE A REPONSE IMPULSIONNELLE NON LINEAIRE

V.3.1. Extraction du modèle à réponse impulsionnelle non linéaire

Les figures ci-dessous présentent les résultats d'extraction de la réponse indicielle (en partie réelle et imaginaire) obtenus par simulation en transitoire d'enveloppe.

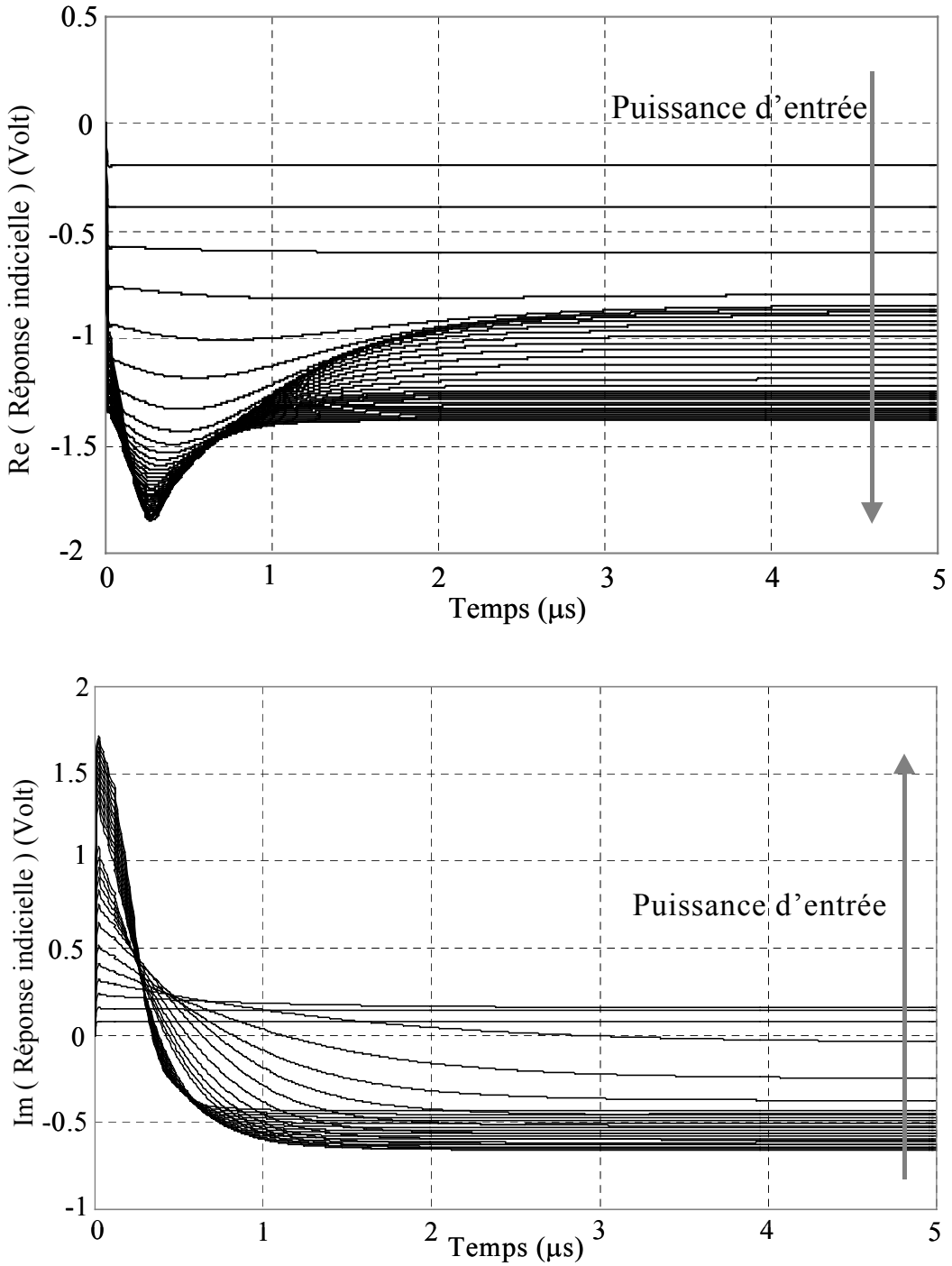


Figure II-37 : Partie réelle/imaginaire de la réponse indicielle obtenue par simulation

Ces deux figures montrent le fonctionnement en régime transitoire de l'amplificateur sur une durée de l'ordre de 5 µs. Nous constatons deux zones de fonctionnement différentes : la première entre 0 et 15 ns, où les variations temporelles de la réponse indicielle sont rapides ; la seconde zone dans laquelle les variations sont plus lentes. Ces deux zones peuvent être respectivement associées à la mémoire HF et BF de l'amplificateur.

Nous présentons sur les figures suivantes la réponse impulsionnelle non linéaire obtenue après simulation et traitement approprié.

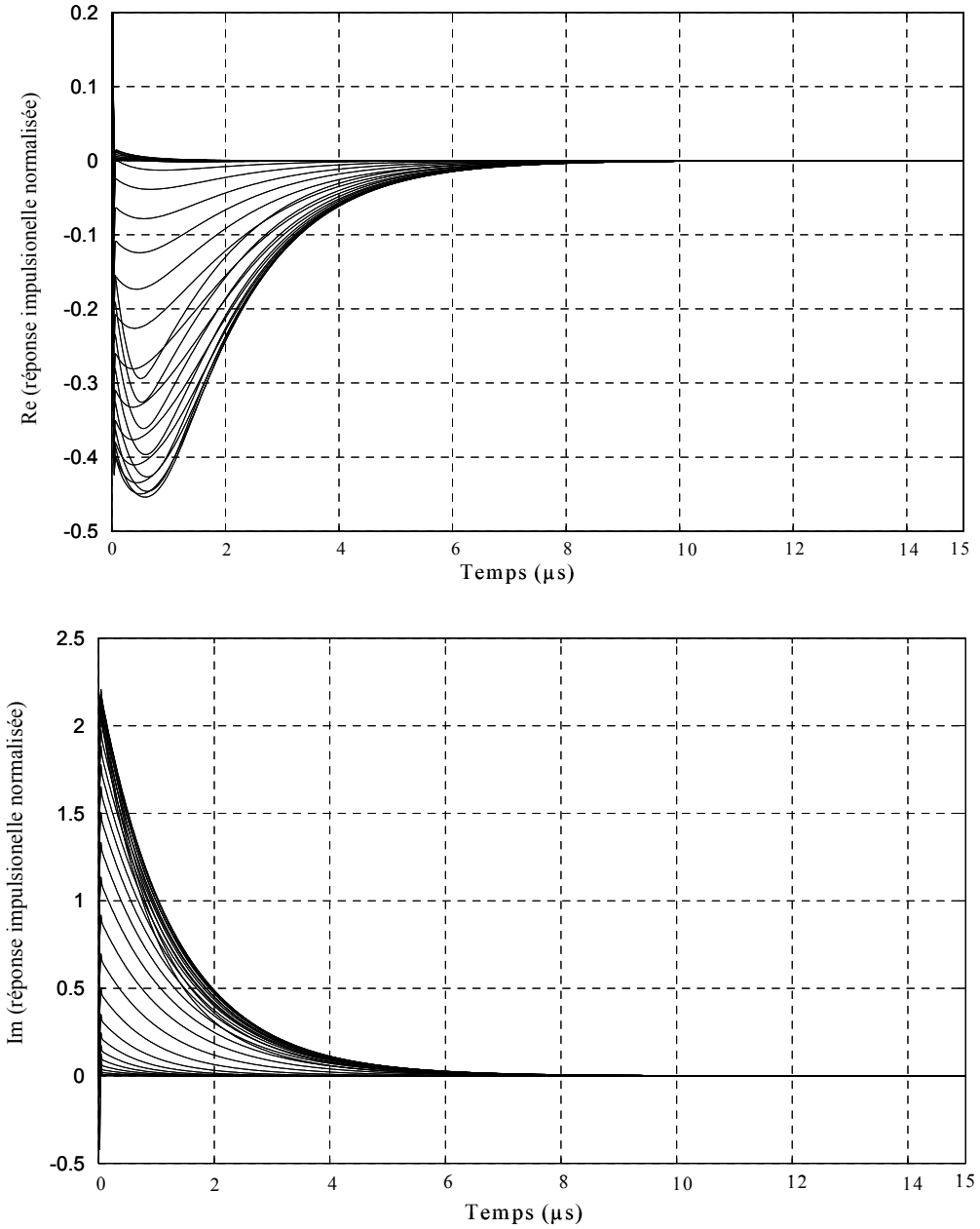


Figure II-38 : Partie réelle/imaginaire de la réponse impulsionnelle normalisée

V.3.2. Validation du modèle

La validation du modèle à réponse impulsionnelle non linéaire implémenté sous Scicos et Simulink a été réalisée en comparant la réponse indicielle obtenue par simulation circuit de celle évaluée par le modèle simulé (figure II-39 et II-40). Nous avons donc excité l'entrée du modèle par un échelon d'Heaviside balayé en puissance afin de couvrir toute la plage de fonctionnement du dispositif.

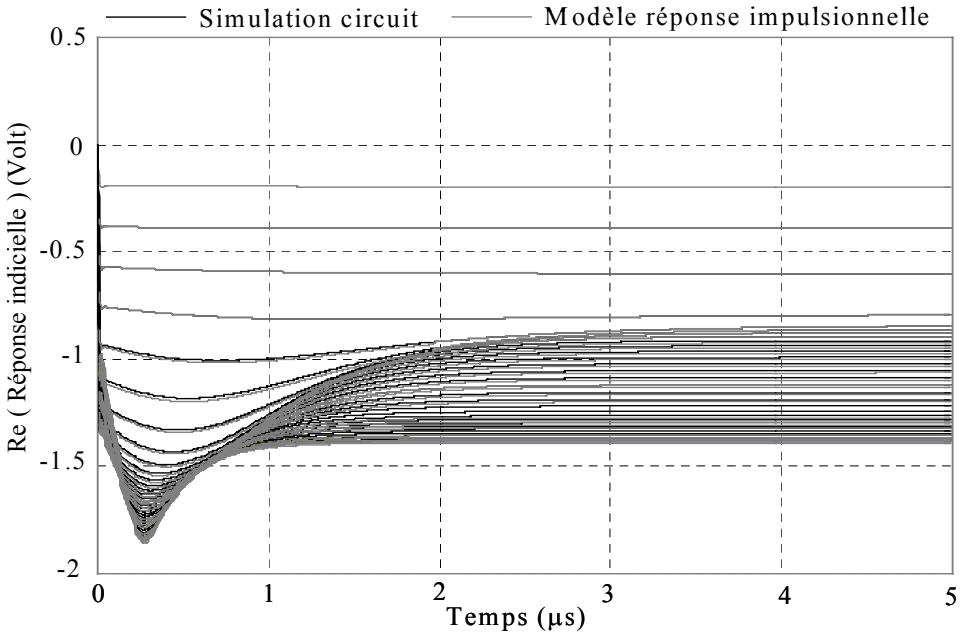


Figure II-39 : Partie réelle de la réponse indicielle obtenue par simulation

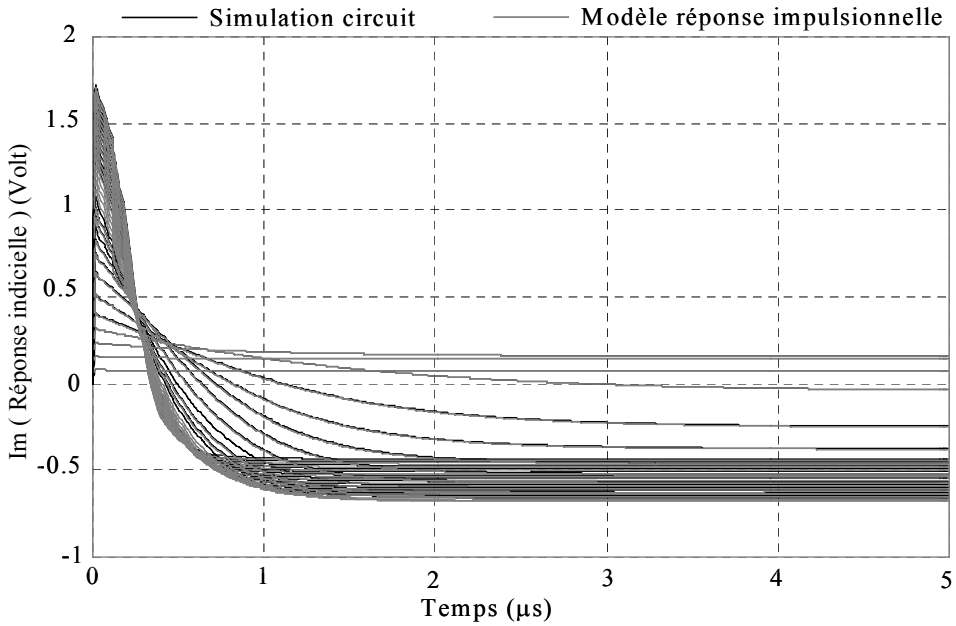


Figure II-40 : Partie imaginaire de la réponse indicielle obtenue par simulation

La réponse indicielle obtenue à partir du modèle laisse apparaître une bonne concordance avec celle dérivée de la simulation circuit.

V.3.3. Intégration des modèles dans une chaîne de communication sous Simulink

Comme nous avons fait pour le modèle de Volterra à deux noyau, nous avons testé les deux modèles dans une chaîne de communication sous Simulink, il s'agit d'un système émetteur récepteur d'une liaison descendante (satellite - station de base). Cette chaîne synthétise le comportement d'une liaison dont le signal d'information est un signal de modulation 16-QAM d'un débit de 5 MHz porté par une porteuse à 8 GHz (figure II-41). Cette chaîne intègre un amplificateur de puissance modélisé par le modèle de Saleh [II-28], le modèle de Volterra à un noyau et le modèle à réponse impulsionnelle non linéaire de l'amplificateur faible bruit sont placés après l'antenne de réception.

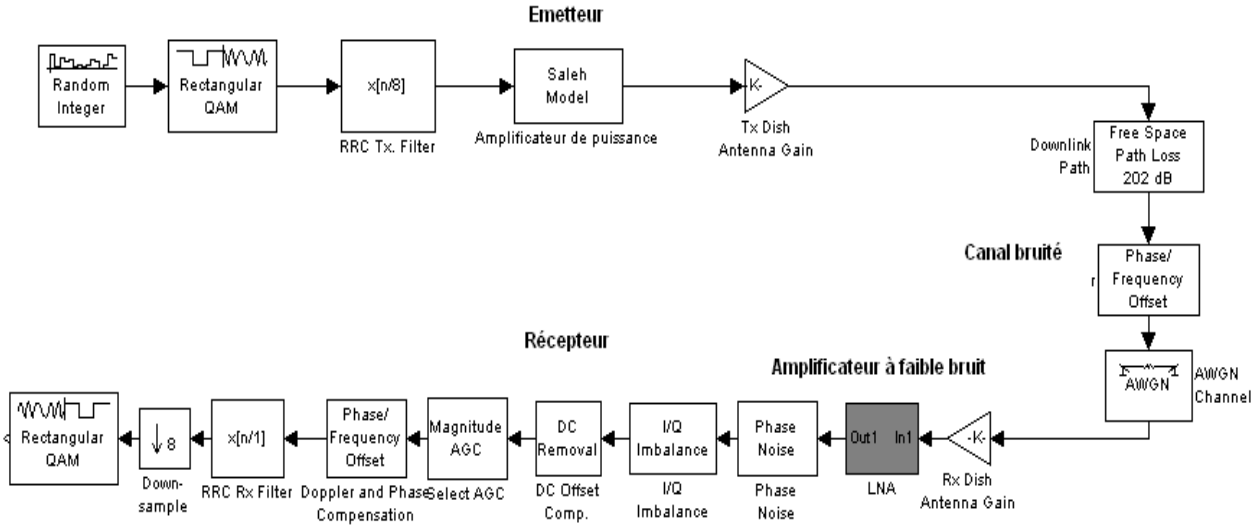
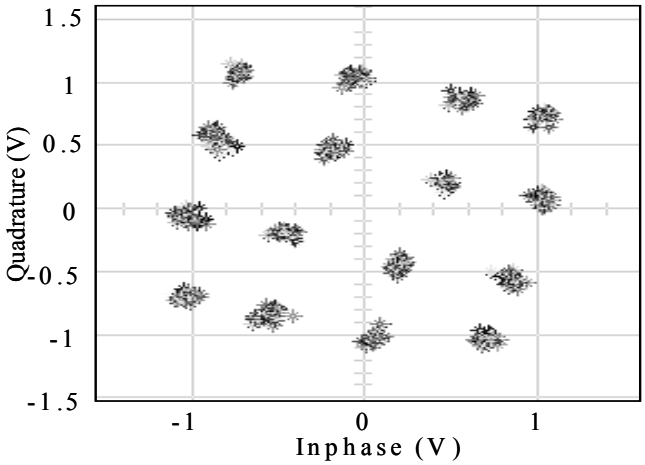
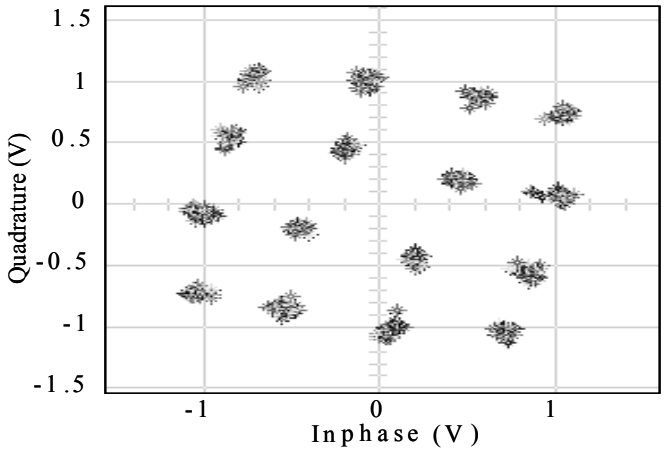


Figure II-41 : Chaîne de communication (satellite - station de base)

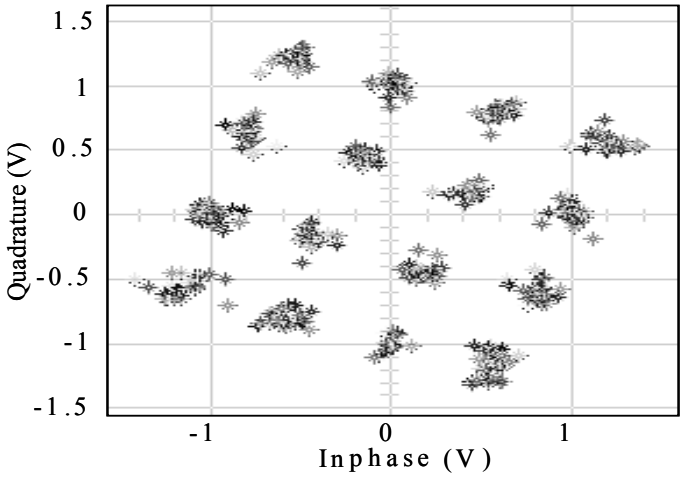
Nous avons effectué la simulation de cette chaîne de communication et les figures suivantes présentent les constellations en réception pour trois niveaux de modélisation de l'amplificateur faible bruit : 1 – en utilisant un modèle sans mémoire, 2 – en utilisant le modèle de Volterra à un noyau et 3 – en utilisant le modèle à réponse impulsionnelle non linéaire.



(a)



(b)



(c)

Figure II-42 : Constellations observées en réception pour un signal 16-QAM de débit 5 MB/s

(a) Modèle sans mémoire, (b) Modèle de Volterra, (c) Modèle à réponse impulsionnelle.

On peut observer sur cet exemple qu'il est essentiel de prendre en compte les phénomènes de mémoire dans les dispositifs non linéaires, pour une estimation du bilan de liaison, notamment en terme d'intégrité de l'information. En particulier, ici, on peut voir que le fait de prendre en compte les phénomènes de mémoire à long terme (c) modifie sensiblement le résultat. Cela se justifie dans notre exemple par le fait que l'amplificateur faible bruit présente des effets basse fréquence importants stimulés par la modulation 16-QAM à 5 MHz [II-29].

Dans le chapitre III, nous allons tester les performances de ces deux modèles dans l'environnement système et comparer ces performances avec la « simulation système» en intégrant le simulateur circuit GoldenGate dans l'environnement Simulink.

VI. Conclusion

Nous avons dans ce chapitre présenté le principe général de deux simulateurs Scicos et Simulink utilisés dans le cadre de notre travail. Ces deux simulateurs nous ont parmi de tester les trois modèles de Volterra et de les intégrer dans des chaînes de communication.

Nous avons présenté également une méthode d'implémentation des modèles de Volterra à deux et à un noyau et le modèle à réponse impulsionnelle non linéaire dans l'environnement Scicos et Simulink. Cette méthode basée sur un développement en série de fonctions des caractéristiques non linéaires permet de conserver à la fois précision de modélisation et rapidité de simulation. Les tests effectués sur les amplificateurs au niveau circuit montre que le modèle de Volterra un noyau permet de prendre les effets mémoire haute fréquence, approprié aux applications à enveloppe quasi-constante. Le modèle à réponse impulsionnelle non linéaire permet de prendre en compte les effets mémoire basse fréquence, dans le cas de signaux à enveloppe fortement variable.

VII. Références

[II-1] **C. Bunks, J. P. Chancelier, F. Delebecque, M. Goursat, R. Nikoukhah and al**
« Engineering and Scientific Computing with Scilab », Birkhauser, 1999.

[II-2] **J. P. Chancelier , F. Delebecque and C. Gomez and al**
« An introduction to Scilab », Springer-Verlag, 2002.

[II-3] **R. Nikoukhah, S. Steer**
« Scicos - A Dynamic System Builder and Simulator », User's Guide, version 1.

[II-4] **R. Nikoukhah**
« Scicos: a dynamic systems modeler and simulator », The 23rd IASTED International Conference on Modeling, Identification, and Control, Grindelwald, Switzerland, February 2004.

[II-5] **M. Najafi, R. Nikoukhah, S. L. Campbell**
« ODE and DAE solvers in Scicos environment », The IASTED International Conference On Applied Simulation And Modelling, Rhodes, Greece, June 2004.

[II-6] **A. Azil, M. Najafi and R. Nikoukhah**
« La génération de code dans Scicos: le cas temps-continu », Réunion Française des Utilisateurs de Logiciels dédiés à la Modélisation et au Calcul Scientifique, Paris, France, Novembre 2004.

[II-7] **R. Djenidi, C. Lavarenne, R. Nikoukhah, Y. Sorel and S. Steer**
« From hybrid system simulation to real-time Implementation », ESS'99 11th European Simulation Symposium and Exhibition, Erlangen-Nuremberg, October 1999.

[II-8] **R. Nikoukhah, S. Steer**
« Scicos : A hybrid system formalism », ESS'99 11th European Simulation Symposium and Exhibition, Erlangen-Nuremberg, October 1999.

[II-9] **R. Nikoukhah, S. Steer**
« Scicos a dynamic system builder and simulator », IEEE international conference on CACSD, Dearborn, Michigan, 1996.

[II-10] R. Djenidi, R. Nikoukhah and S. Steer
« A propos du formalisme Scicos », MOSIM'01, Avril 2001, Troyes, France.

[II-11] M. Najafi, A. Azil, R. Nikoukhah
« Implementation of Continuous-Time Dynamics in Scicos », 15th European Simulation Symposium and Exhibition, Delft, Netherlands, October 2003.

[II-12] M. Najafi, A. Azil, R. Nikoukhah
« Extending Scicos from system to component level Simulation », ESMc2004, international conference, Paris, France, October 2004.

[II-13] The MathWorks Inc
« Simulink : Dynamic System Simulation for MATLAB », Using Simulink, version 6.

[II-14] The MathWorks Inc
« Simulink : Writing S–Functions »,Using Simulink, version 6.

[II-15] S. Le Ballois
«Matlab/Simulink Application à l'automatique linéaire », Paris : Ellipses, 2001.

[II-16] M. Mokhtari, A. Mesbah
« Apprendre et maîtriser MATLAB », Springer-Verlag, France, 1997.

[II-17] L. Shampine, M. Reichlet
« The Matlab ODE Suite », SIAM J. Sci. Computing, pages 1-22, 1997.

[II-18] R. Ashino, M. Nagase, R. Vaillancourt
« Behind and Beyond the Matlab ODE Suite », CRM, pages 491–512 January 2000.

[II-19] D. Fredon, J. Ezquerra, M-C. Chatard-Moulin, A. Salinier
« Les Fondamentaux. Analyses », Hachette Education Paris, 1995.

[II-20] J. Bartkovjak , M. Karovičová
« Approximation by Rational Functions », Measurement science review, Volume 1, Number 1, 2001.

[II-21] M. Schatzman
« Analyse Numérique», Inter Editions,1991.

[II-22] C.W. Gear

« Numerical initial value problems in ordinary differential equation », Prentice-Hall Inc, 1971.

[II-23] J. Vlach, K. Singhal

« Computer Methods for Circuit Analysis and Design », Van Nostrand Reinhold, 2nd edition, 1994.

[II-24] D.B. Kuznetsov, J.E. Schutt-Ainé

« The Optimal Transient Simulation of Transmission Lines », IEEE Trans. Circuits Syst.-I., vol.n°43, pp. 110-121, February 1996.

[II-25] A. Bennadji, A. Soury, S.Mons, E. Ngoya, R.Quéré

« Implementation of dynamic Volterra model in Scicos Environment », 1st Scilab International Conference, Paris, Janvier 2004.

[II-26] A. Bennadji, A. Soury and E. Ngoya

« Implementation of dynamic Volterra model in system Environment », International Workshop on Electronics and System Analysis, Bilbao Espagne, Octobre 2004.

[II-27] A. Layec

« Développement de modèles de C.A.O. pour la simulation système des systèmes de communication. Application aux communications chaotiques», Thèse de Doctorat de l'université de Limoges, Février 2006.

[II-28] A. Saleh

« Frequency Independent and Frequency dependant Nonlinear models of TWT Amplifiers », IEEE Transactions on communications, vol. com-29, no. 11, pp.1715-1719, November 1981.

[II-29] A. Bennadji, A. Soury, E. Ngoya, R. Quéré

« Implémentation des modèles comportementaux dans des environnements de simulation système », 14èmes Journées Nationales Microondes (JNM 2005), Nantes, France, Mai, 2005.

CHAPITRE III

co-simulation circuit système

I. Introduction

Dans le chapitre précédant, nous avons présenté la méthode d'implémentation des modèles d'amplificateurs basés sur les séries de Volterra dans les environnements de simulation système généraux tels que Matlab/Simulink™ et Scilab/Scicos™. Ces modèles se sont avérés plus ou moins précis pour tous les types de signaux et l'extraction de ces modèles est accessible à partir des simulateurs de circuit et les bancs de mesures. Dans un effort d'améliorer la précision de la modélisation, nous avons développé une interface de co-simulation qui permet au simulateur de circuit d'accéder à un simulateur de système pour chaque échantillon. Dans notre exemple, nous avons considéré la co-simulation entre Matlab/Simulink™, Scilab/Scicos™ et le simulateur circuit Xpedion/GoldenGate™.

Dans ce chapitre, nous aborderons les différents mécanismes de communication inter-processus qui nous permettent d'échanger les données entre le simulateur circuit et système. Nous présenterons les mécanismes de synchronisation entre les deux simulateurs, pour garantir l'intégrité des informations transférées. Ces mécanismes sont basés sur la programmation système API Win32, il s'agit d'une interface de programmation d'applications API (Application Programming Interface) pour les applications Windows. À la fin du chapitre, nous présenterons les résultats de co-simulation obtenus à partir de ce travail.

II. Intégration d'un simulateur circuit dans un simulateur système

Dans ce paragraphe, nous allons aborder les mécanismes d'intégration du simulateur circuit dans un simulateur système. Considérons par exemple la chaîne de communication présentée sous Scicos ou Simulink (Figure III-1), nous voulons réaliser la co-simulation à l'interface entrée/sortie de l'amplificateur, pour cela il nous faut implémenter l'interface de co-simulation dans les deux simulateurs (système et circuit). Au niveau du simulateur système, l'interface de co-simulation est à réaliser dans un bloc *Généric* sous Scicos et dans un bloc C MEX S-Function sous Simulink. Au niveau du simulateur circuit, nous avons implémenté cette interface dans le noyau du simulateur. Ainsi, le bloc d'amplificateur sous le simulateur système représente une instance d'un simulateur circuit, qui réalise l'analyse de transitoire d'enveloppe. A chaque activation de bloc, les tâches suivantes sont réalisées :

- Lancement du simulateur circuit pour l'exécution de la simulation d'enveloppe du circuit, cette simulation peut contenir plusieurs pas d'analyse ;
- Retourner les résultats de simulation au simulateur système.

Dans cette configuration, le simulateur circuit est conçu comme processus fils du simulateur système. Dans la suite de ce chapitre, nous allons présenter la configuration père/fils qui va nous permettre d'intégrer le simulateur circuit dans le simulateur système et de réaliser l'échange des données entre les deux simulateurs.

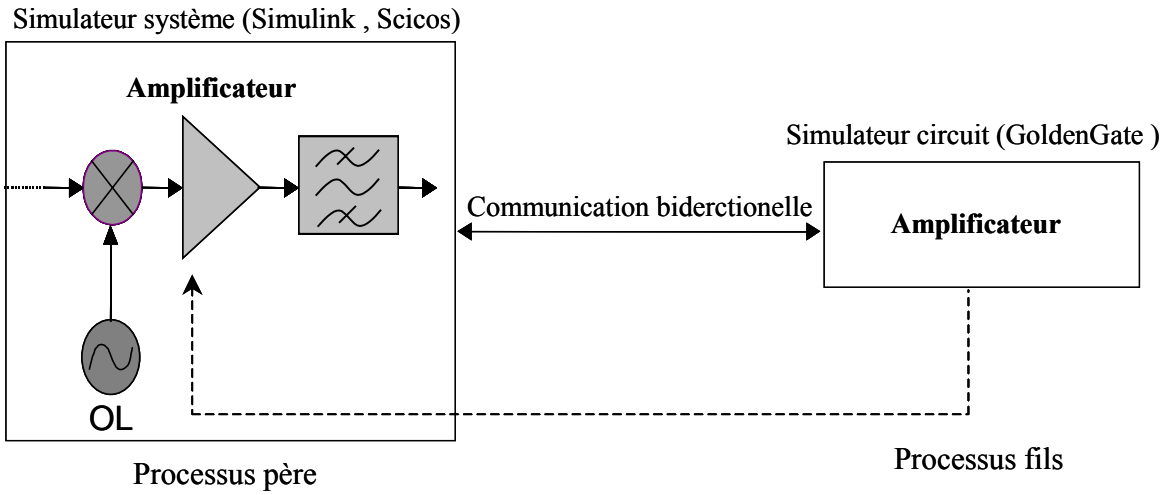


Figure III-1 : Principe de la co-simulation

II.1. L'ARCHITECTURE PERE/FILS

La communication entre les différents simulateurs est un point essentiel dans la définition d'un environnement de co-simulation. Les simulateurs sont exécutés en parallèle et l'environnement de co-simulation doit assurer l'intégrité des données (c-à-d pas d'inversion ou d'écrasement de données). Ainsi, il est nécessaire que l'environnement de co-simulation dispose d'un modèle de synchronisation fiable et performant en vitesse.

Nous avons choisi dans ce travail de thèse le modèle de communication basé sur l'architecture père/fils [III-1], ce modèle désigne de quelle façon les simulateurs échangent leurs données à travers l'environnement de co-simulation. La vitesse et l'efficacité de la co-simulation dépendent du modèle de synchronisation et des moyens de communication utilisés.

Le modèle père/fils part du principe qu'il existe un seul simulateur père et un ou plusieurs simulateurs fils concernés dans le co-simulation. La figure III-2 présente un modèle générique de co-simulation père/fils. La réalisation de procédure de communication entre les deux simulateurs est basée sur des mécanismes standards du système d'exploitation, tels que les IPC dans le cas du système Windows. La communication entre le père et le fils se fait de la façon suivante :

- 1. Attendre l'arrivée d'une requête émise par le père ;
- 2. Exécuter la requête du père ;
- 3. Retourner la réponse au père qui a émis la requête ;
- 4. Retourner à la phase 1.

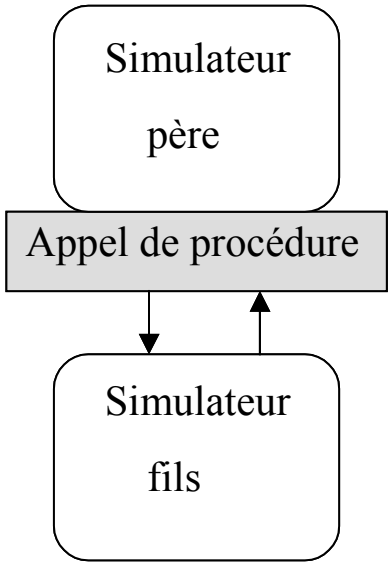


Figure III-2 : Modèle générique de la co-simulation père/fils

La figure III-3 représente le mécanisme d'échange de données entre les deux simulateurs selon le modèle de synchronisation père/fils. Les échanges en lecteur/écriture entre les deux simulateurs se font de façon alternée. Le simulateur fils (simulateur circuit) attend l'arrivée des évènements d'entrée qui proviennent du simulateur père (simulateur système). Le simulateur père est arrêté pendant l'exécution du simulateur fils en attendant les évènements générés par ce dernier. Ce modèle de synchronisation ne permet pas d'exécuter les simulateurs de façon parallèle [III-2].

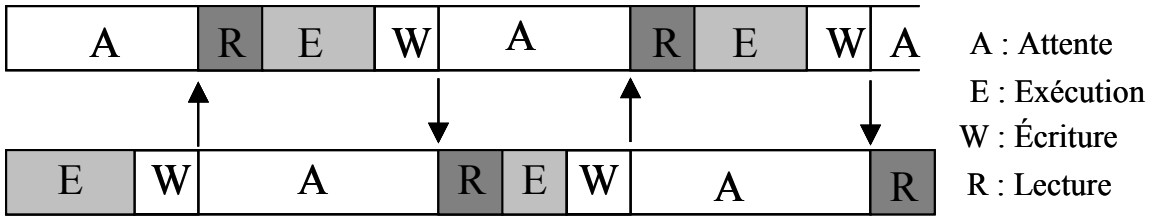


Figure III-3 : Synchronisation père/fils

II.1.1. Interface coté simulateur circuit

Le bloc que nous avons utilisé pour intégrer le simulateur circuit dans le simulateur système est le générateur de tension de type IQ, la figure III-4 schématise un bloc GoldenGate implémenté dans le simulateur système qui possède deux ports d'entrée et deux ports de sortie.

Nous avons implémenté dans les deux simulateurs (Scicos et Simulink), trois types de blocs (amplificateurs et mélangeurs) :

- Un bloc avec deux ports d'entrée et deux ports de sortie (Figure III-4) ;
- Un bloc avec deux ports d'entrée et quatre ports de sortie ;
- Un bloc avec quatre ports d'entrée et deux ports de sorties.

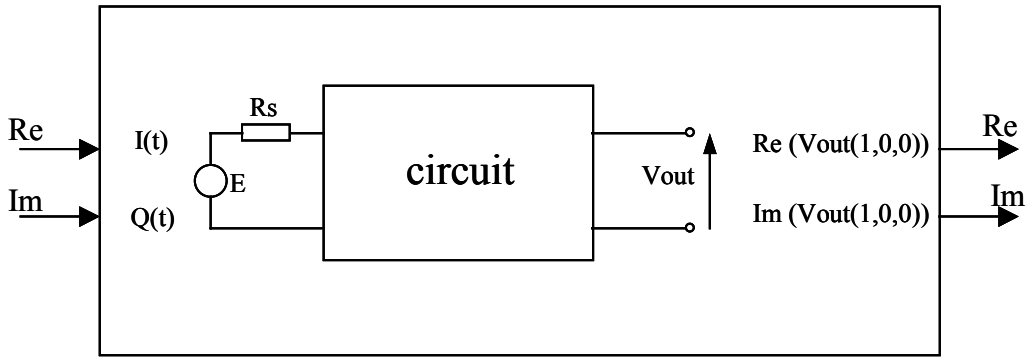


Figure III-4 : Le bloc GoldenGate implémenté dans le simulateur système

La tension E dans le générateur est donnée par :

$$E(t) = Re\left[\left(I(t) + jQ(t)\right)e^{j\omega_0 t}\right] \quad \text{(III-1)}$$

Où $I(t)$ et $Q(t)$ représentent respectivement le signal d'entrée du bloc correspondant aux ports un et deux et ω_0 la fréquence porteuse. La valeur de ω_0 ainsi que la résistance interne de chaque générateur sont définies dans le circuit.

Les sorties du bloc GoldenGate sont mesurées par des sondes du circuit, desquelles on récupère les parties réelles et imaginaires du signal d'enveloppe.

II.1.2. Interface coté simulateur système

Nous avons utilisé les mêmes blocs qui étaient utilisés pour implémenter les modèles comportementaux dans les deux environnements de simulation système (Simulink et Scicos). Dans le simulateur Simulink, nous avons utilisé les blocs S-Functions et les blocs Generic dans le simulateur Scicos.

La figure ci-dessous montre la bibliothèque que nous avons implémenté dans le simulateur Simulink, cette bibliothèque contient les trois types de bloc que nous avons implémenté.

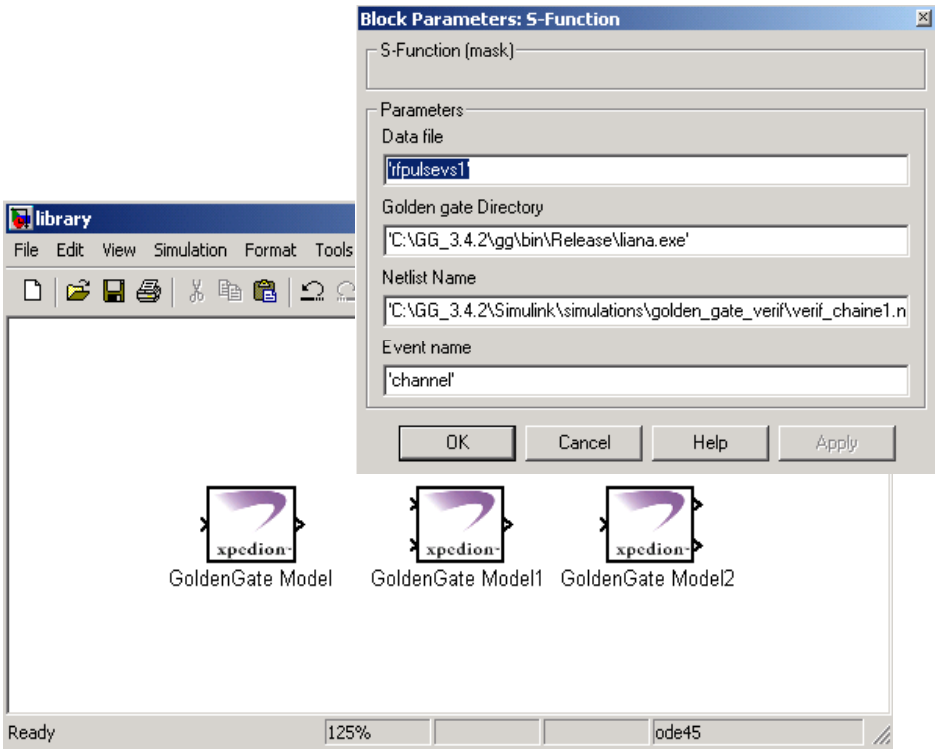


Figure III-5 : Bibliothèque de co-simulation dans Simulink

III. Communication entre deux simulateurs

III.1. PRESENTATION GENERALE DE LA GESTION DE PROCESSUS

Avant de montrer les différents mécanismes de communication inter-processus, nous commençons, tout d'abord, par définir le processus et les méthodes de gestion du processus.

III.1.1. Le processus

Un processus désigne une application en cours d'exécution. Il y a autant de processus actifs que d'application en cours d'exécution. Si on lance trois fois de suite une même application, Windows crée trois processus.

Pour Windows, les processus s'exécutent dans des espaces mémoires indépendants. Un processus ne peut pas accéder à la zone mémoire d'un autre processus, même s'il s'agit de deux processus d'une même application [III-3].

Un processus sous Windows est caractérisé par deux valeurs : un handle (de type HANDLE) et un identifiant (de type DWORD). Le handle permet au processus appelant « *père* » d'envoyer des messages au processus créé « *fils* ». L'identifiant permet au processus appelant « *père* » d'atteindre un processus créé « *fils* » (par exemple pour fermer un processus, …).

III.1.2. Création d'un processus

La création d'un processus se fait par l'appel de la routine *CreateProcess()* de l'API Win32. A l'appel de cette fonction, le noyau crée un objet noyau processus. Cet objet n'est pas le processus à proprement parler mais une structure de données permettant la gestion de ce processus. Le noyau crée ensuite l'espace d'adressage et y recopie le code exécutable et les données du processus. Puis le système crée un objet noyau thread qui correspond aux structures de données du thread principal associé au processus. Ce thread commence son exécution au début du code exécutable.

III.1.2.1. Prototype de la fonction CreateProcess

La fonction CreateProcess crée dynamiquement un nouveau processus et son thread initial. Le nouveau processus exécute le fichier exécutable spécifié en argument. La création d'un processus ne peut se faire sans lui associer un code exécutable.

```
BOOL CreateProcess (
                    LPCTSTR lpApplicationName
                    LPTSTR lpcommandLine,
                    LPSBCURITY_ATTRIBUTES IpProcessAttributes,
                    LPSECURITY_ATTRIBUTES lpThreadAttributes,
                    BOOL blnherithandles,
                    DWORD dwCreationFlags,
                    LPVOID lpEnvironment,
                    LPCTSTR lpCurrentDirectory,
                    LPSTARTUPNFO lpStartuplnfo,
                    LPPROCESS_INFORMATION IpProcessinformation
                    );
```

Cette fonction doit retourner `TRUE` seulement si le processus a été créé avec succès.

Voici un exemple d'appel à la fonction *CreateProcess()* :

```
CreateProcess(
              "liana.exe",             // Nom du module
              "liana.exe -stdout -wait -ipc1 1 0 0",// Nom de la commande
              NULL, // Attributs de securite du process par defaut
              NULL, // Attributs de securite du thread par defaut
              TRUE,          // Heritage des handles du createur
              CREATE_SUSPENDED, // Etat du thread principal
              NULL,          // Environnement du createur
              NULL,          // Repertoire du createur
              &startup_info, // Informations sur la fenêtre
              &process_info // Récupération d'une structure PROCESS_INFO
              );
```

Le nom du module (1er paramètre) pointe sur une chaîne de caractères représentant le chemin d'accès complet du programme exécutable à lancer, (2ème paramètre) correspond à la ligne de commande avec tous les paramètres ; dans notre cas, nous mettons l'exécutable du simulateur circuit GoldenGate. Nous avons remarqué la possibilité de lancer le simulateur circuit GoldenGate en utilisant comme ligne de commande `"liana.exe -stdout -wait -ipc1 1 0 0"`. Le simulateur circuit GoldenGate est alors correctement lancé mais il reçoit une ligne de commande avec laquelle il n'aurait jamais pu être lancé. Le fait que la plupart des programmes n'utilisent jamais l'argument 0 (nom de la commande) rend une telle erreur difficile à détecter.

Le 3ème paramètre pointe sur une structure `SECURITY_ATTRIBUTES` contenant les attributs de sécurité du processus créé.

Le 4ème paramètre pointe sur une structure `SECURITY_ATTRIBUTES` contenant les attributs de sécurité liés au thread initial.

Le 5ème paramètre permet au processus créé d'hériter des handles du processus appelant, si ce paramètre vaut `TRUE`, tous les handles ouverts du processus appelant, sont hérités par le nouveau processus créé.

Le 6ème paramètre indique le mode de création du processus. Ici l'exécution du thread principal est suspendue dès sa création. Il ne sera activé que lors de l'appel de la fonction *ResumeThread()* par le processus créateur.

Le 7ème paramètre pointe sur un bloc contenant les variables d'environnement pour le nouveau processus, si ce paramètre vaut `NULL`, le nouveau processus utilise l'environnement du processus appelant.

Le 8ème paramètre pointe sur une chaîne de caractères indiquant au nouveau processus le répertoire courant, si ce paramètre vaut `NULL`, le nouveau processus utilise le répertoire courant du processus appelant. Le 9ème paramètre pointe sur une structure *startup_info*.

Le nouveau processus créé s'exécute indépendamment du processus appelant et ne partage pas le même espace d'adresses virtuelles. Lorsque la fonction s'exécute avec succès, une structure *process_info* (10ème et dernier paramètre) est retournée, contenant le handle et l'identificateur du processus et de son thread principal, cette structure est comme suit :

```
typedef struct PROCESS_INFORMATION {
                                    HANDLE hProcess;
                                    HANDLE hThread;
                                    DWORD dwProcessId;
                                    DWORD dwThreadId;
                                    }PROCESS_INFORMATION;
```

`HANDLE hProcess`, handle sur le nouveau processus créé ;

`HANDLE hThread`, handle sur le premier thread du processus créé ;

`DWORD dwProcessId`, identificateur global du processus créé ;

`DWORD dwThreadId`, identificateur global du premier thread du processus créé.

III.1.3. Identifier un processus

C'est généralement à travers une valeur de handle qu'un processus créé est identifié de façon unique par le processus appelant. Lorsqu'un processus est créé avec *Createprocess()*, deux types de handles différents sont retournés. Dans les deux cas, les handles sont retournés avec les droits d'accès complets, et sont donc sujets aux mécanismes de vérification de la sécurité d'accès. Ces handles peuvent être utilisés sans aucune restriction dans toutes les fonctions qui attendent ce genre de paramètres. Suivant les attributs qui définissent les conditions d'héritage,

ces handles peuvent ou non être hérité par le processus créé, et restent valides tant qu'ils n'ont pas été fermés par *CloseHandle()* (handle d'objet index dans une table d'objet spécifique à chaque objet, il fait référence à l'objet ouvert et contient un ensemble de droits d'accès accordés au processus qui le possède).

Pour obtenir le handle d'un processus à partir de son identifieur, nous utilisons la fonction *OpenProcess()*, cette fonction est utile pour gérer l'attente d'un processus par un autre et pour terminer le processus.

```
HANDLE OpenProcess(
                    DWORD dwDesiredAccess,
                    BOOL bInheritHandle,
                    DWORD dwProcessId
                    );
```

`DWORD dwDesiredAccess`, flag d'accès (exemples : `PROCESS_ALL_ACCESS`, `PROCESS_TERMINATE`) ;

`BOOL bInheritHandle`, flag d'héritage des handles ;

`DWORD dwProcessId`, identificateur du processus.

III.1.4. La terminaison d'un processus

Une fois que le processus est créé, il reste chargé en mémoire et s'exécute jusqu'à ce que survienne un des événements suivants :

- Un des threads du processus invoque *ExitProcess()* ;
- Le thread principal retourne ;
- L'exécution du dernier thread se termine ;
- *TerminateProcess()* est invoqué.

Les opérations effectuées sont les suivantes :

- Tous les handles ouverts sont fermés, les objets correspondants ne sont pas obligatoirement détruits (d'autres processus peuvent avoir ouvert des handles sur les mêmes objets) ;
- L'état de l'objet processus devient signalé (ceci peut permettre de synchroniser des threads en attente de cet événement).

Pour terminer le processus spécifié et tous ses threads, nous utilisons la fonction *TerminateProcess()*, cette fonction retourne une valeur non nulle, elle retourne 0 en cas d'échec.

```
BOOL TerminateProcess(
                        HANDLE hProcess,
                        UINT uExitCode
                        );
```

`HANDLE hProcess`, identifie le processus à terminer ;

`UINT uExitCode`, code de retour du processus.

III.1.5. Exemple de code

La gestion de processus fils (simulateur circuit) est gérée dans le processus père (simulateur système), voici un exemple du code implémenté dans le simulateur Simulink et Scicos.

```
#include <windows.h>
if( !CreateProcess(NULL,
                    command_line,
                    NULL,
                    NULL,
                    FALSE,
                    CREATE_NEW_CONSOLE,
                    NULL,
                    NULL,
                    &si,
                    &pi ))
           {
              return false;
           }
process_id = OpenProcess(PROCESS_ALL_ACCESS, TRUE,pi.dwProcessId);
TerminateProcess(process_id, 0);
```

`command_line` correspond au chemin et les paramètres du simulateur circuit GoldenGate.

III.2. PRESENTATION DU MECANISME DE COMMUNICATION INTER-PROCESSUS

Comme nous l'avons évoqué dans le paragraphe précédant, le dialogue entre le simulateur système et le simulateur circuit repose sur le mécanisme de communication IPC (**I**nter **P**rocess **C**ommunication) [III-4]. Ce mécanisme vise à rendre transparente, pour le simulateur système comme pour le simulateur circuit, l'échange des données entre les deux simulateurs.

Ce paragraphe présente de manière succincte les principales méthodes de communication inter-processus utilisées sur les plates-formes Windows. L'étude de ces IPC n'étant pas le but de cette partie, nous nous limiterons ici à l'approche de leur principe de fonctionnement, en évitant autant que possible les détails d'implémentation.

III.2.1. Fichiers

La communication par fichier (ASCII ou Binaire) repose sur la capacité qu'ont les différents processus d'accéder au système de fichiers local. Son principe de base est le suivant (figure III-6) :

1. Le processus père copie les données à transférer dans un fichier au moyen des primitives standards d'accès aux fichiers ;

2. Le processus cible récupère (par une autre méthode IPC ou par une convention commune) le nom et le chemin d'accès du fichier. Il peut ensuite lire et exploiter les données contenues dans le fichier (ce principe est facilement étendu au cas de lecteurs multiples).

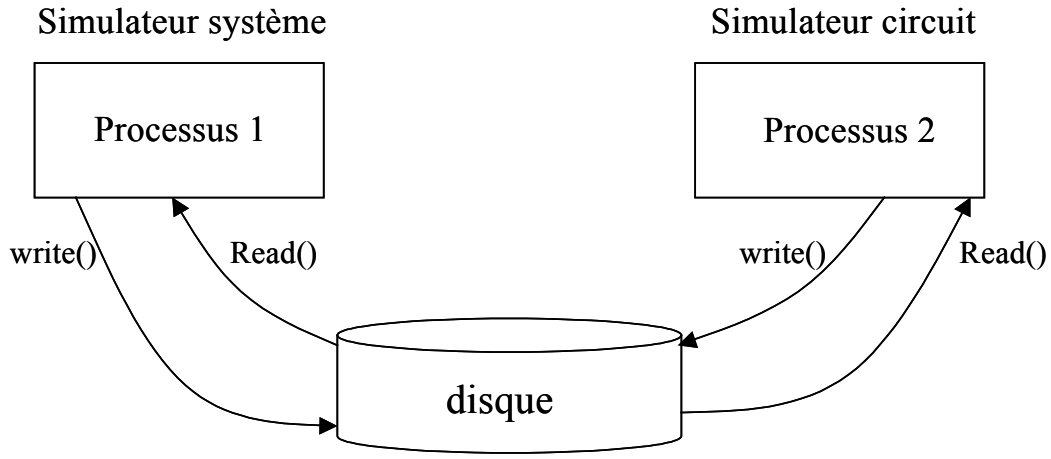


Figure III-6 : IPC par fichier standard

Cette méthode a l'avantage d'être très simple à mettre en oeuvre, de supporter des transferts de très gros volumes de données (à priori, la taille des fichiers est illimitée) et d'être utilisable même dans le cas où les deux processus s'exécutent à des périodes de temps disjointes. Néanmoins, cette méthode est rarement utilisée pour les transferts entre deux processus, en effet, outre le fait que le passage par le disque introduit une lourde pénalité au niveau des performances, il revient aussi aux processus la tâche de synchroniser leurs accès au fichier.

III.2.2. ***Pipes***

Les fichiers ne fournissent pas un moyen satisfaisant de communication entre les processus car le fichier garde trace de toutes les informations transmises, ce qui pose des problèmes, en particulier de place disque.

Les pipes (tubes) [III-5] fournissent un mécanisme bien adapté à la communication inter-processus, deux types de pipes coexistent sur les systèmes Windows : les tubes anonymes (*anonymous pipes*) et les tubes nommés (*named pipes*). Les pipes anonymes sont plus simples à mettre en oeuvre, mais n'autorisent pas toutes les possibilités des tubes nommés. Dans les deux cas, l'idée sous-jacente est toujours de construire un canal de communication entre deux (ou plusieurs) applications, qui soit accessible comme un simple fichier local.

III.2.2.1. Pipes anonymes

Un pipe anonyme est un conduit d'information unidirectionnel utilisable entre deux applications possédant un lien de parenté. Il n'est donc employable que dans un contexte local et ne peut pas être utilisé à travers un réseau. Le lien de parenté est requis pour le transfert des handles identifiant le pipe (un pour l'écriture et un pour la lecture). Pour réaliser une communication bidirectionnelle entre les deux simulateurs, il faut utiliser deux pipes anonymes un pour communiquer entre le simulateur système et le simulateur circuit et l'autre entre le simulateur circuit et le simulateur système (Figure III-7). L'accès au pipe s'effectue au moyen de fonctions de lecture/écriture *(ReadFile(..)* et *WriteFile(..*)).

Les pipes anonymes peuvent être utilisés uniquement en environnement local, c'est-à-dire qu'ils ne peuvent pas être mis en œuvre pour assurer une communication à travers un réseau.

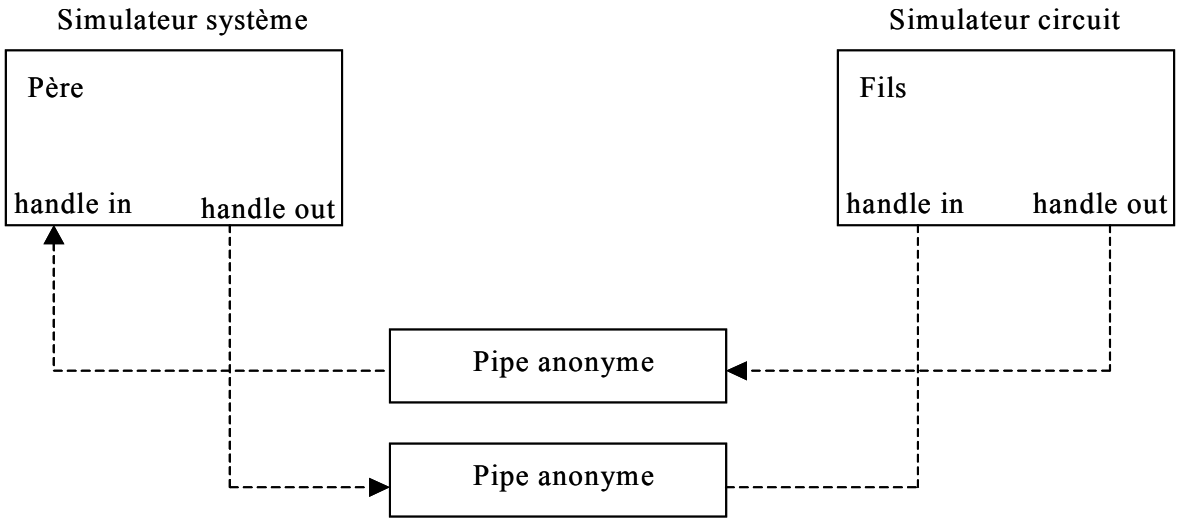


Figure III-7 : Communication entre simulateurs par les pipes anonymes

III.2.2.2. Pipes nommés

Un pipe nommé est un canal de communication bidirectionnel entre une application (le pipe père) et un certain nombre d'applications fils. Toutes les instances d'un pipe nommé partagent le même nom, mais possèdent chacune leur propre buffer et leur propre handle, ce qui leur permet de fournir un canal de communication père-fils séparé à chaque père (Figure III-8). Cette particularité permet au fils d'identifier aisément l'émetteur d'un bloc de données et évite de devoir multiplier les pipes sur le système.

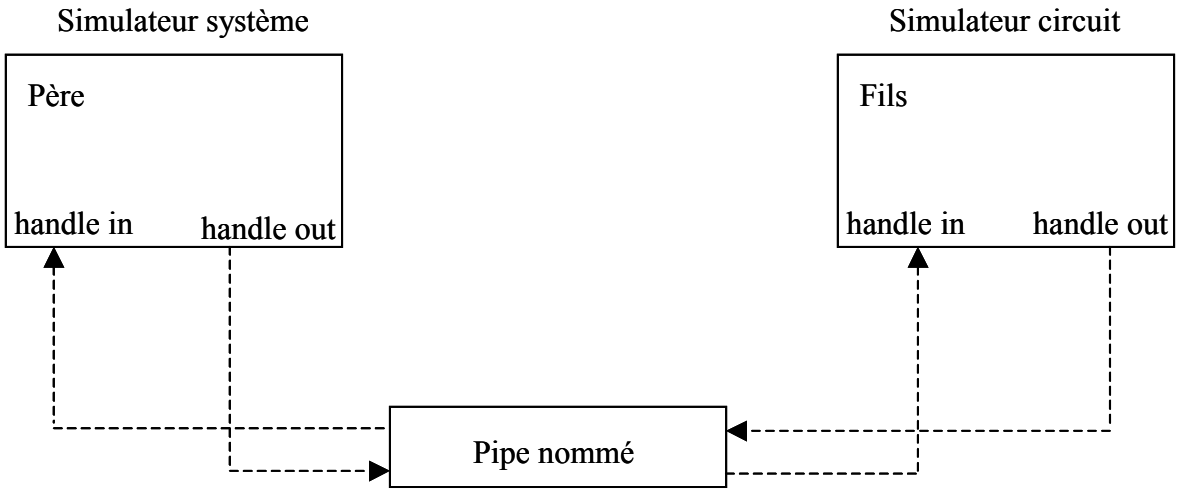


Figure III-8 : Communication entre simulateurs par le pipe nommé

III.2.3. Mailslots (Boîte à lettres)

Un mailslot [III-6] est un pseudo-fichier : il réside entièrement en mémoire centrale mais est accessible au moyen des fonctions d'accès aux fichiers standards (*WriteFile()* et *ReadFile()*). Les données placées dans un mailslot peuvent prendre n'importe quelle forme tant que la taille totale ne dépasse pas 64 Ko. Contrairement aux fichiers standards, les mailslots sont temporaires : lorsque tous les handles vers un mailslot sont fermés, ce dernier est détruit et les données qu'il contenait sont perdues.

Les mailslots ont la particularité de pouvoir être utilisés comme moyens de communication distribués : si un client distant obtient un handle sur un mailslot, les données qu'il va y lire et écrire sont transmises via le réseau. Les messages de moins de 425 bytes sont transmis au moyen d'un datagramme UDP, les messages de taille supérieure sont transmis via une connexion TCP dans le cadre d'une session SMB.

Le principe de fonctionnement d'un mailslot est celui d'une file FIFO : chaque application possédant le handle du mailslot peut y ajouter un message (tout en respectant la limite de taille globale de 64K), lire un message ou retirer le premier message de la liste.

La faible capacité (un maximum de 64 Ko) de cette IPC la relègue à des tâches très spécifiques où sa capacité de distribution sur l'ensemble du domaine local et sa relative simplicité d'utilisation sont des avantages par rapport aux autres méthodes, généralement beaucoup plus lourdes à mettre en oeuvre dans le cas distribué.

III.2.4. Mémoire Partagée

L'idée sous-jacente à l'IPC par mémoire partagée [III-7] est de fournir aux processus une zone de mémoire commune à laquelle ils peuvent librement accéder et de les laisser gérer eux-mêmes les éventuels problèmes de concurrence (Figure III-9). Il s'agit là du strict minimum pour une IPC, le noyau donne accès à une zone utilisable par tous les processus participants et laisse le programmeur libre d'en réglementer ou non l'accès. Les adresses en mémoire sont dites virtuelles, ce qui signifie par exemple que l'adresse 1792 dans le processus père, ne correspond pas au même emplacement physique que l'adresse 1792 du processus fils. De cette manière, un processus n'a aucun moyen d'accéder aux données d'un autre processus.

Le partage de données entre deux processus différents se fait, dans Windows, à l'aide des fichiers mappés (mapped files). Un fichier mappé est un espace d'adressage virtuel auquel plusieurs processus peuvent accéder simultanément. La fonction *CreateFileMaping()* crée un

fichier mappé dont la taille peut être importante. Lors de l'appel de cette fonction, il n'y a pas d'allocation de mémoire physique, mais simplement la mise en réserve d'un ensemble d'adresses virtuelles. Ce n'est que lors de l'appel de la fonction *MapViewOfFile()* qu'une partie de la mémoire réservée est allouée et que les processus peuvent y accéder. Au moment de sa création avec la fonction *CreateFileMaping()*, le fichier mappé reçoit un nom, comme les autres processus connaissent ce nom, ils peuvent y accéder. Une autre utilisation des fichiers mappés permet d'accéder à un fichier comme s'il s'agissait d'une zone située en mémoire.

Donc pour créer une mémoire partagée entre les deux simulateurs, il faut dans un premier temps créer un objet noyau fichier mappé dans le simulateur système en utilisant la fonction *CreateFileMaping()*. Les objets fichiers mappés peuvent être nommés, ce qui permet leur partage. Le prototype de cette fonction est comme suit :

```
HANDLE CreateFileMapping (
                          HANDLE hFile,
                          LPSECURITY_ATTRIBUTES lpsa,
                          DWORD fdwProtect,
                          DWORD dwMaximumSizeHigh,
                          DWORD dwMaximumSizeLow,
                          LPTSIR lpszMapName
                          );
```

`HANDLE hFile`, indique le handle du fichier à partir duquel nous voulons créer un objet mappé sur le disque, si `hFile` est `INVALID_HANDLE_VALUE` ou `(HANDLE)0xFFFFFFFF`, nous n'utilisons pas un fichier spécifié sur le disque mais un fichier d'échange système ou "system swap file" ;

`LPSECURITY_ATTRIBUTES` lpsa, pointeur sur les attributs de sécurité, si lpsa est `NULL` le fichier obtient un descripteur de sécurité par défaut ;

`DWORD fdwProtect`, indique le type de protection pour la vue sur le fichier, si `fdwProtect` est `PAGE_READONLY`, nous ne pouvons pas écrire dans l'espace mémoire, si `fdwProtect` est `PAGE_READWRITE`, nous pouvons lire et écrire dans l'espace mémoire ;

`DWORD dwMaximumSizeHigh`, le poids fort de la taille de l'objet section ;

`DWORD dwMaximumSizeLow`, le poids faible de la taille de l'objet section ;

`LPTSIR lpszMapName`, nom de l'objet fichier mappé, si `NULL`, le handle retourné permet de l'identifier.

Après la création de l'objet noyau fichier mappé, il faut mapper les données de ce fichier dans l'espace d'adressage virtuel du processus avec la fonction *MapViewOfFile()*. Le prototype de cette fonction est comme suit :

```
LPVOID MapViewOfFile (
                      HANDLE hMapObject,
                      DWORD fdwAccess,
                      DWORD dwOffSetHigh,
                      DWORD dwOffSetLow,
                      DWORD cbMap
                      );
```

La valeur de retour indique l'adresse de début de la vue mappée, elle sera utilisée par les fonctions de lecture et d'écriture.

`HANDLE hMapObject`, la valeur de retour du handle de l'objet fichier mappé créé par *CreateFileMapping()* ;

`DWORD fdwAccess`, indique le type de protection comme `fdwProtect` ;

`DWORD dwOffSetHigh`, représente l'offset (32 bits poids forts) dans le fichier à partir duquel doit commencer le mapping de la vue ;

`DWORD dwOffSetLow`, représente l'offset (32 bits poids faibles) dans le fichier à partir duquel doit commencer le mapping de la vue ;

`DWORD cbMap`, le nombre d'octets à mapper, si `cbMap` égale à 0, alors la totalité du fichier est mappé.

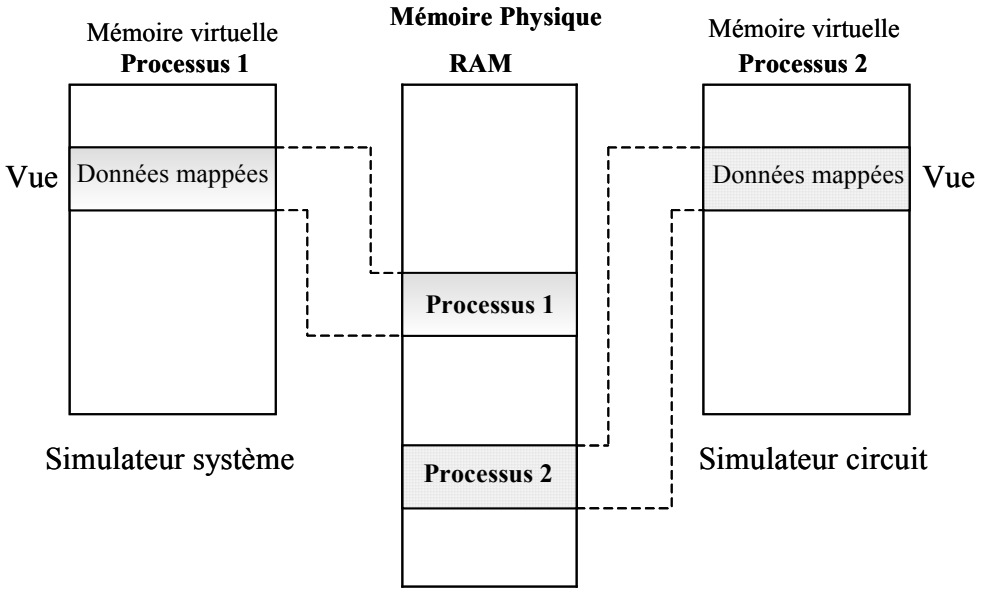


Figure III-9 : Principe de la mémoire partagée

Pour que le simulateur circuit puisse se rattacher à ce segment de mémoire partagée, il faut dans un premier temps obtenir un handle sur l'objet fichier mappé en passant son nom en paramètre à la fonction *OpenFileMapping()*. Le prototype de cette fonction est comme suit :

```
HANDLE OpenFileMapping (
                         DWORD dwDesiredAccess,
                         BOOL bInheritHandle,
                         LPTSTR lpName
                         );
```

La fonction retourne `NULL` en cas d'échec.

`DWORD dwDesiredAccess`, défini le type d'accès à la section, cet argument doit être compatible avec `fdwProtect` de la fonction *CreateFileMaping()* ;

`BOOL bInheritHandle`, si `TRUE` le handle retourné est hérité par le processus fils lors de l'appel à *CreateProcess()* par le père ;

`LPTSTR lpName`, pointeur sur la chaîne de caractère qui nomme l'objet section.

Le simulateur circuit doit aussi utiliser la fonction *MapViewOfFile()* pour mapper les données du fichier dans l'espace d'adressage virtuel du processus fils.

Enfin, à la fin de la simulation, le processus père doit invoquer la fonction *UnmapViewOfFile()*. Cette fonction détruit la vue sur le fichier mappé au sein de son espace d'adressage. Le prototype de cette fonction est comme suit :

```
BOOL UnmapViewOfFile (
                       LPVOID lpBaseAddress
                      );
```

La fonction retourne `FALSE` en cas d'échec, sinon `TRUE`.

`LPVOID lpBaseAddress`, valeur de retour de la fonction *MapViewOfFile()*.

III.2.5. Exemple de code

Voici un exemple de code implémenté dans le simulateur système (Simulink et Scicos) et le simulateur circuit GoldenGate.

Le simulateur système :

```
#include <windows.h>
shared_mem_ptr = CreateFileMapping(INVALID_HANDLE_VALUE,
                                   NULL,
                                   PAGE_READWRITE,
                                   0,
                                   map_size,
                                   file_name);
if(shared_mem_ptr==NULL)return false;
open_shared_mem= OpenFileMapping(FILE_MAP_ALL_ACCESS,
                                 TRUE,
                                 file_name);
memory_ptr = (char* ) MapViewOfFile(open_shared_mem,
                                    FILE_MAP_ALL_ACCESS,
                                    0,
                                    0,
                                    0);
UnmapViewOfFile(memory_ptr);
CloseHandle(open_shared_mem);
```

Le simulateur circuit :

```
#include <windows.h>
open_shared_mem= OpenFileMapping(FILE_MAP_ALL_ACCESS,
                                 TRUE,
                                 file);
memory_ptr = (char* ) MapViewOfFile(open_shared_mem,
                                    FILE_MAP_ALL_ACCESS,
                                    0,
                                    0,
                                    0);
```

III.2.6. Classification des différentes méthodes

Nous allons tenter, dans ce paragraphe, de faire une classification des différentes IPC présentées dans le paragraphe précédant. L'idée sous-jacente est de montrer qu'il n'existe pas de meilleure IPC dans l'absolu, mais simplement des IPC plus ou moins adaptées à certains contextes de communication. Nous allons évaluer les IPC sur la base des critères suivants : portée, protection, parallélisme et débit.

III.2.6.1. Portée

Par portée d'une IPC, nous entendons l'ensemble des processus capables d'entrer en communication avec un processus donné au moyen de cette IPC. Les deux cas de figure les plus courants sont la portée globale, où tous les processus du système peuvent communiquer avec le processus de référence et la portée locale, où seuls les processus possédant une relation de filiation plus ou moins directe avec le processus de référence sont en mesure de communiquer avec lui.

III.2.6.2. Protection

Le critère de protection consiste à évaluer dans quelle mesure les circonstances exceptionnelles survenant lors de la communication sont à la charge des processus participants. Il conviendra en particulier de prendre en compte la protection contre les accès concurrents ainsi que la protection contre les accès non autorisés.

III.2.6.3. Parallélisme

Le critère de parallélisme consiste à voir si l'IPC analysée supporte que la communication contienne plusieurs processus père/processus fils à la place d'un seul couple (père/fils). Ensuite, il s'agira de voir si, dans la situation de père/fils multiples, chaque processus participant peut clairement identifier les messages qui lui sont destinés sans pour autant devoir recourir à une méthode externe où à l'analyse de tous les messages.

III.2.6.4. Débit

Ce critère prend en compte les performances des transferts de données (le délai minimum nécessaire pour le transfert d'une information simple et la capacité de l'IPC à soutenir des transferts de gros volumes d'information).

IPC	Portée	Protection	Parallélisme	Débit
Fichier	Importante	Faible	Moyen	Faible
Pipes anonymes	Faible	Faible	Faible	Moyen
Pipes nommés	Importante	Faible	Important	Moyen
Mailslots	Faible	Faible	Faible	Faible
Mémoire partagée	Importante	Moyenne	Important	Important

Tableau III-1 : Synthèse des éléments de comparaison

Nous remarquons sur le tableau III-1 que la mémoire partagée présente de meilleurs avantages par rapport aux autres mécanismes IPC. Le mécanisme de la mémoire partagée est le plus difficile à implémenter (car tous les cas de figures doivent être envisagés) mais aussi le plus puissant en terme de performances et de parallélisme. C'est pour cette raison que nous avons utilisé ce mécanisme pour communiquer entre les deux simulateurs [III-8].

III.3. SYNCHRONISATION DES SIMULATEURS

Toute interaction entre les deux simulateurs requiert la prise en charge des aléas engendrés par leurs exécutions parallèles : la plupart des ressources nécessaires à la communication entre les deux simulateurs doivent être protégées contre les tentatives d'accès concurrents afin de garantir l'intégrité des informations transférées. Pour cela, il faut mettre en place le mécanisme de synchronisation [III-9] pour éviter les conflits lors de partages de données entre les deux simulateurs, Windows met à la disposition des développeurs quatre objets de synchronisation :

- Les objets événements ;
- Les objets mutants (mutex ou exclusion mutuelle) ;
- Les objets section critique ;
- Les objets sémaphores.

Un objet se trouve obligatoirement dans l'un des deux états suivants : signalé ou non signalé. La synchronisation ne s'effectue que du passage de l'état non signalé vers l'état signalé. Par exemple, les objets processus passent à l'état signalé lors de la terminaison du processus qu'ils désignent. Le tableau III-2 présente la définition des quatre types d'objets de synchronisation.

Event (évènement)	Pour signaler un évènement ou indiquer qu'une condition ou un ensemble de condition est vrai.
Mutex (mutant)	Objet qui permet d'accéder en exclusion mutuelle à une zone de données partagées entre plusieurs threads ou pour se protéger d'une exécution simultanée d'une procédure. Un Mutex est possédé par un seul thread à la fois.
Critical Section (section critique)	Objet équivalent au Mutex, la différence réside dans le fait que les sections critiques sont réservées aux threads d'un même process. Il est plus simple à utiliser et son implémentation a été optimisée.
Sémaphore	Objet qui permet de contrôler le nombre de threads pouvant accéder simultanément à une ressource partagée (version à compte du mutex).

Tableau III-2 : Les quatre types d'objets de synchronisation

Dans ce travail de thèse, nous avons utilisé les objets événements. Dans le paragraphe, suivant, nous allons détailler le principe de base de ce mécanisme.

III.3.1. Les objets événements

Les objets événements [III-10] sont à assimiler aux signaux sous Unix. Windows permet de créer autant d'événements que l'utilisateur le souhaite contrairement à Unix où le nombre de signaux est fixe.

Il existe deux types d'objets événements :

- Les événements à réinitialisation manuelle : le passage de l'état signalé à l'état non signalé doit être effectué explicitement par un appel à la fonction *ResetEvent()* ;
- Les événements à réinitialisation automatique : le passage de l'état signalé à l'état non signalé s'effectue automatiquement dès qu'un processus en attente de cet événement a pris connaissance de son état signalé.

Le type de l'événement est déterminé à sa création lors de l'appel de la fonction *CreateEvent()*. La syntaxe de cette fonction est présentée comme ci-dessous :

```
HANDLE CreateEvent (
                    LPSECURITY_ATTRIBUTES lpEventAttributes,
                    BOOL bManualReset,
                    BOOL bInitialState,
                    LPCTSTR lpName
                    );
```

`LPSECURITY_ATTRIBUTES lpEventAttributes`, pointeur sur les attributs de sécurité `SECURJTY_ATTRIBUTS` ;

`BOOL bManualReset`, si `TRUE` il s'agit d'un signal manuel, sinon un signal automatique ; Dans le premier cas il faut utiliser la fonction *ResetEvent()* pour ramener l'objet Event à un état non signalé.

`BOOL bInitialState`, si `TRUE` l'état initial de l'événement est signalé, sinon non signalé;

`LPCTSTR lpName`, nom de l'objet événement. Il est limité à `MAX_PATH` caractères et peut contenir n'importe quel caractère sauf *backslash*. Si `NULL`, alors l'objet est anonyme.

L'objet événement peut être utilisé par plusieurs processus (le processus appelant et le processus créé). Si nous créons un objet évènement dans le processus appelant, dans ce cas le

processus créé doit invoquer *OpenEvent()* afin de récupérer un handle sur cet objet événement déjà crée. La fonction *OpenEvent()* retourne un handle sur un Objet Evénement nommé.

```
HANDLE OpenEvent(
                 DWORD dwDesiredAccess,
                 BOOL bInheritHandle,
                 LPCTSTR lpName
                );
```

`DWORD dwDesiredAccess`, flag d'accès;

`BOOL bInheritHandle`, flag d'héritage (si `TRUE`, les processus créés par le processus appelant héritent du handle) ;

`LPCTSTR lpName`, pointeur sur le nom de l'objet événement.

Un événement passe de l'état non signalé à l'état signalé en utilisant la méthode *SetEvent()*. Cette fonction permet d'envoyer un signal de déclenchement de processus appelant vers le processus créé et vice-versa. Dans notre cas, nous avons utilisé cette fonction dans le simulateur système pour lancer le simulateur circuit et dans le simulateur circuit pour laisser la main au simulateur système quand la première étape de simulation circuit est terminée. La syntaxe de cette fonction est comme suit :

```
BOOL SetEvent(
              HANDLE hEvent,
              );
```

`HANDLE hEvent`, handle sur l'événement spécifié.

Tout processus peut se synchroniser sur un événement. La primitive *WaitForSingleObject()* permet de préciser sur quel objet noyau le processus souhaite se synchroniser. La durée de l'attente de synchronisation peut être infinie ou fixée à l'aide d'une valeur de timeout exprimée en millisecondes. La syntaxe de cette fonction est comme suit :

```
DWORD WaitForSingleObject(
                          HANDLE hHandle,
                          DWORD dwMilliseconds
                          );
```

`HANDLE hHandle`, Handle de l'objet à attendre ;

`DWORD dwMilliseconds`, Timeout en millisecondes (si la valeur est nulle le test est non-bloquant, sinon il s'agit d'un timeout en millisecondes, `INFINITE` pour une attente infinie).

III.3.2. Exemple de code

Nous avons implémenté les objets évènements dans les deux simulateurs pour assurer la synchronisation entre eux. Nous donnons ici un exemple de code pour bien comprendre le mécanisme de synchronisation entre les deux simulateurs.

Le simulateur système :

```
#include <windows.h>
process_event12 = CreateEvent(NULL, FALSE, FALSE,
                              process_name.12);
process_event21 = CreateEvent(NULL, FALSE, FALSE,
                              process_name.21);
SetEvent(process_event12);

WaitForSingleObject(process_event21,INFINITE);
```

Le simulateur circuit :

```
#include <windows.h>
HANDLE  process_event12 = OpenEvent(EVENT_ALL_ACCESS,
                                    FALSE,
                                    process_name.12);
HANDLE  process_event21 = OpenEvent(EVENT_ALL_ACCESS,
                                    FALSE,
                                    process_name.21);
WaitForSingleObject(process_event12,INFINITE);
SetEvent(process_event21);
```

Nous avons créé deux évènements nommés dans le simulateur système (process_event12 et process_event21). Dans le simulateur circuit, nous avons utilisé la fonction *OpenEvent()* afin de récupérer les handles sur les objets événements créés dans le simulateur système. Après la récupération des handles, le simulateur circuit se met en attente de l'évènement process_event12 à l'aide la fonction *WaitForSingleObject()*, le simulateur système envoie cet événement à l'aide de la fonction *SetEvent()* pour lancer la simulation de l'amplificateur. Lorsque le simulateur circuit termine la simulation pour une trame numérique, il envoi l'événement process_event21 pour laisser la main au simulateur système afin de continuer la simulation de la chaîne de communication.

IV. Application

Dans cette dernière partie du chapitre nous allons montrer des exemples de simulation réalisés avec le principe de co-simulation que nous avons décrit dans les paragraphes précédents, mettant en relation les simulateurs système (Simulink et Scicos) et le simulateur circuit GoldenGate.

Le premier exemple présente la chaîne de communication sous Simulink utilisée dans le chapitre II. Nous rappelons que cette chaîne de communication représente un système émetteur récepteur d'une liaison descendante (satellite - station de base), le signal d'information est un signal 16-QAM porté par une porteuse à 8 GHz. La co-simulation se fait au niveau de l'amplificateur à faible bruit (LNA) placé après l'antenne de réception (illustration Figure III-10 pour Simulink et Figure III-11 pour Scicos).

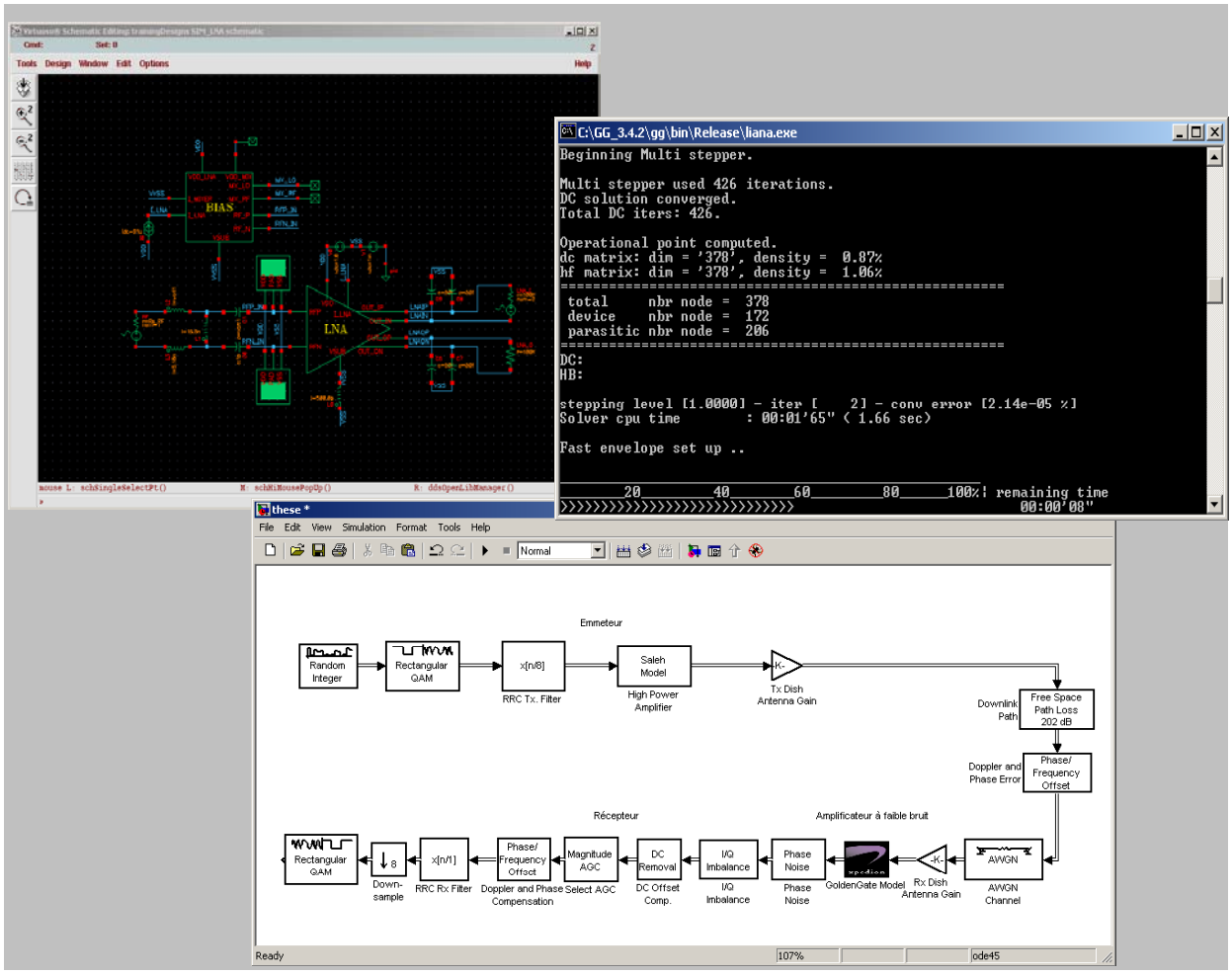


Figure III-10 : La co-simulation entre Simulink et GoldenGate

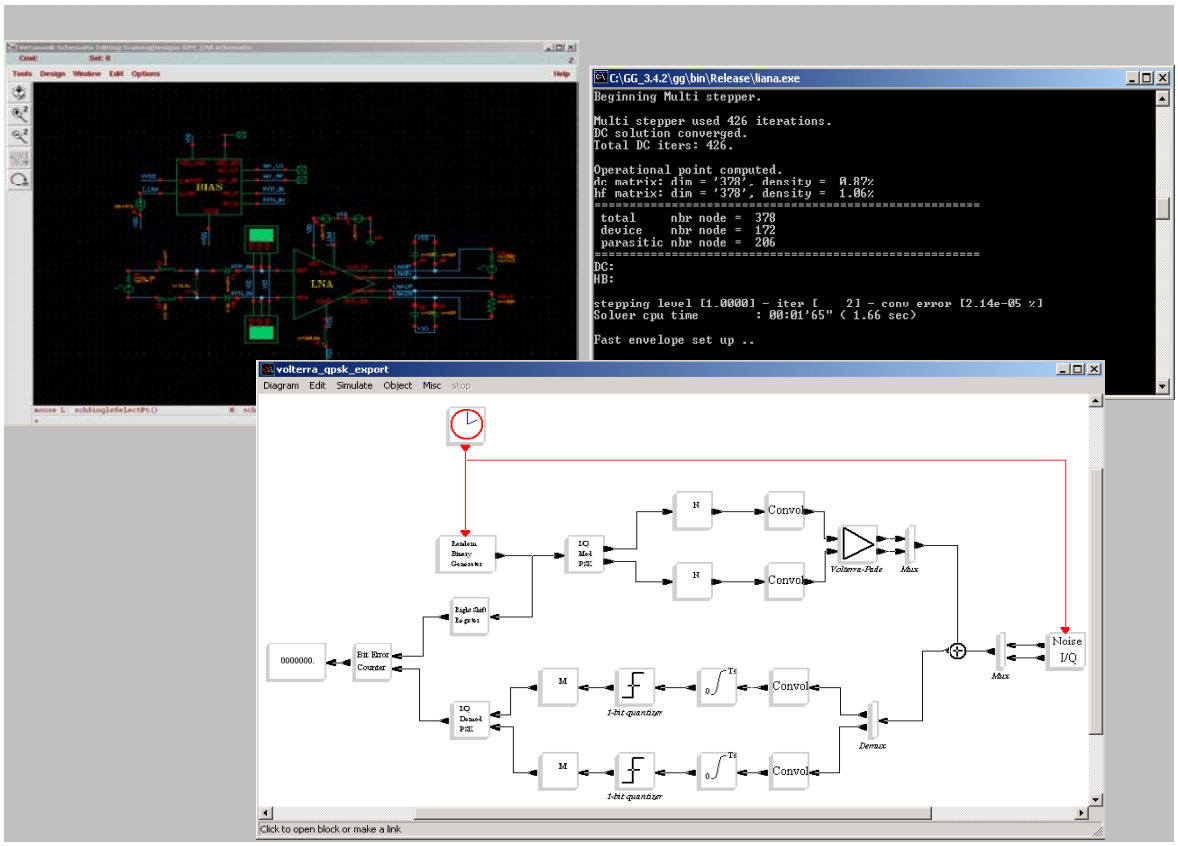


Figure III-11 : La co-simulation entre Scicos et GoldenGate

Il est intéressant de voir le comportement des modèles en présence de signaux de modulation numérique couramment rencontrés. Les performances des modèles ont donc été évaluées en présence d'un signal 16-QAM, filtré par un filtre demi-Nyquist avec un roll-off de 0.35. La figure suivante présente la constellation en réception pour un signal de modulation 16-QAM d'un débit de 5 MHz.

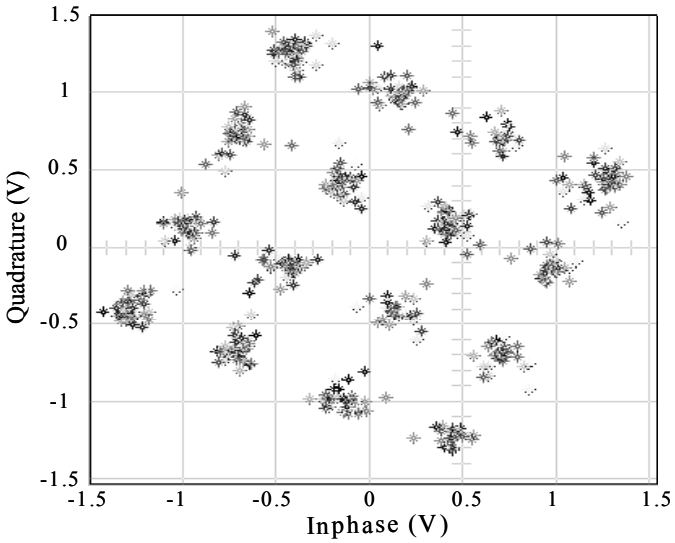


Figure III-12 : Constellation observée en réception pour un signal 16-QAM @ 5 MB/s

Nous observons sur cette figure les distorsions d'amplitude et de phase sur le diagramme de constellation des états, cette perturbation est due principalement à l'utilisation de l'amplificateur à faible bruit (LNA) qui prend en considération les effets de mémoire et la non linéarité.

Nous présentons sur la figure III-13, la comparaison des résultats de l'ACPR obtenus par co-simulation, par le modèle de Volterra à un noyau, par le modèle à réponse impulsionnelle non linéaire et le modèle AM/AM AM/PM dans le cas d'une modulation 16-QAM pour un débit de 5 MHz.

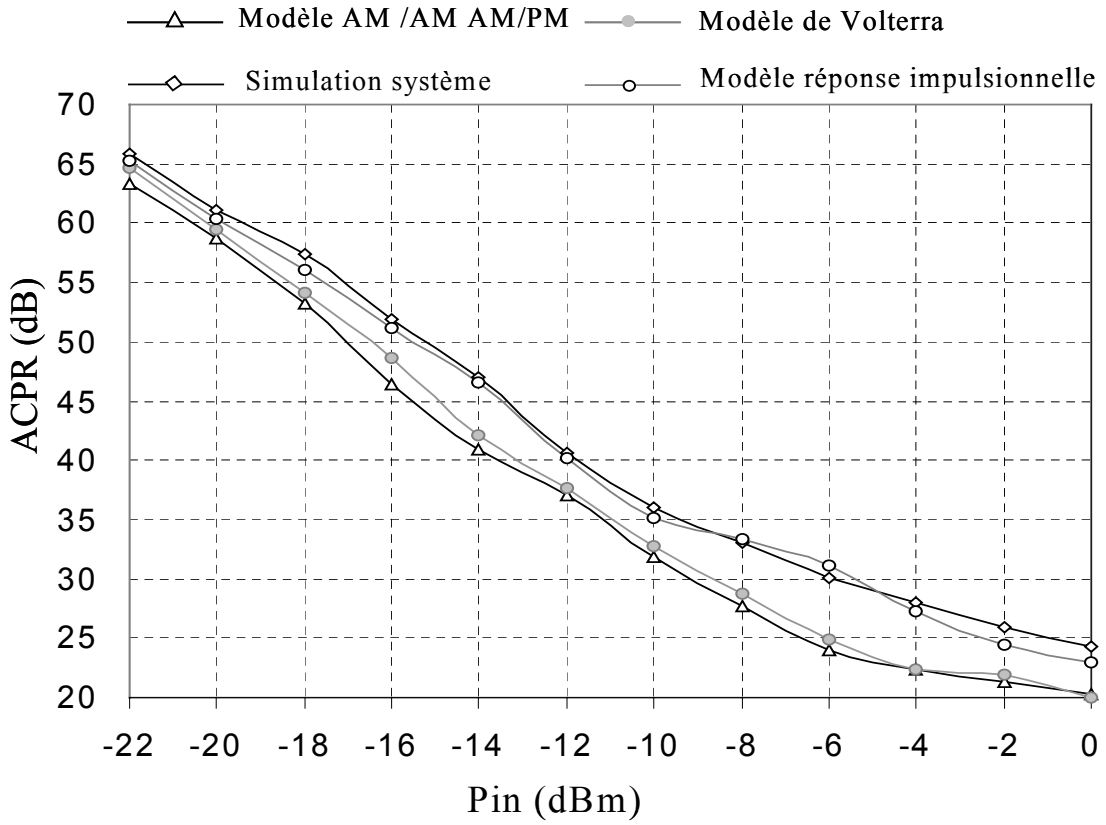


Figure III-13 : ACPR en fonction de la puissance d'entrée (16-QAM @ débit 5 MHz)

Les résultats présentés dans la figure III-13 montrent que l'accord entre le modèle à réponse impulsionnelle non linéaire et le résultat obtenu par co-simulation est assez bon. Le résultat du modèle de Volterra un noyau et le modèle AM/AM AM/PM pour la prédiction du ACPR, reste loin de l'ACPR donné par la co-simulation (4 à 7 dB) du fait d'un impact important de la mémoire basse fréquence pour cet amplificateur.

Le temps de simulation de l'ACPR pour les différents modèles implémentés et la co-simulation pour la chaîne complète est récapitulé dans le tableau III-3. Le temps de simulation

du modèle de Volterra un noyau ainsi que celui du modèle à réponse impulsionnelle non-linéaire est d'environ dix fois supérieur au modèle AM/AM AM/PM. Les performances du modèle à réponse impulsionnelle montrent donc une bonne précision de simulation, en restant dans des temps de simulation convenables.

	Co-simulation	Modèle AM/AM AM/PM	Modèle de Volterra un noyau	Modèle de réponse impulsionnelle
Temps de simulation de la chaîne complète	1h. 12 min.	14 sec.	2 min. 18 sec.	2 min. 23 sec.

Tableau III-3 : Temps de simulation de 16-QAM @ débit 5 MHz

Nous avons tracé également les allures temporelles de l'enveloppe du signal de sortie de l'amplificateur obtenues par co-simulation et par simulation des deux modèles pour des signaux de test GMSK et 16-QAM. La figure III-14 présente les enveloppes temporelles correspondantes à la modulation 16-QAM pour un débit de 5 MHz correspondant aux courbes d'ACPR présentées dans la figure III-13.

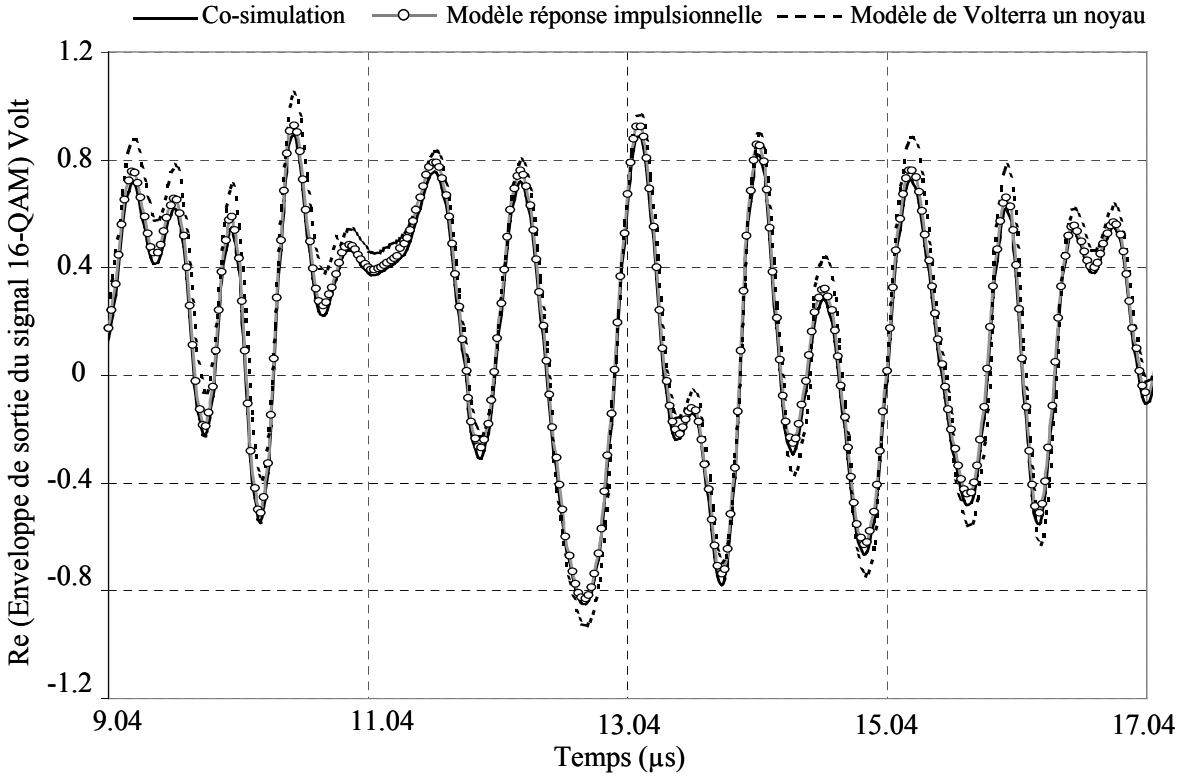


Figure III-14 : Partie réelle de l'enveloppes temporelles d'une 16-QAM @ 5 MHz

On remarque une bonne concordance de forme d'onde entre le modèle à réponse impulsionnelle et la co-simulation, mais par contre des erreurs non négligeables pour le modèle de Volterra à un noyau. Ceci confirme les observations faites à partir des courbes d'ACPR.

Nous avons testé également ces modèles à l'aide du signal 16-QAM pour un débit plus grand, de 50 MHz. La figure ci-dessous présente les résultats d'ACPR pour les deux modèles implémentés et le modèle statique (AM/AM AM/PM) et la co-simulation.

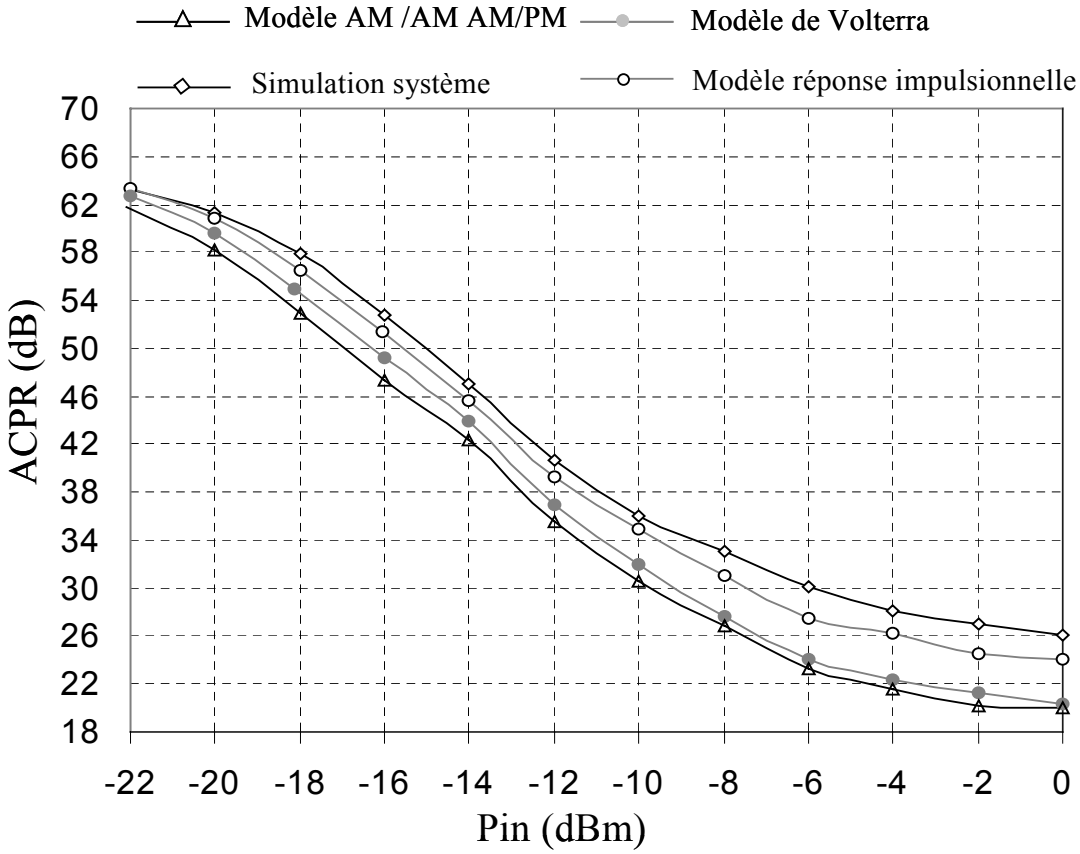


Figure III-15 : ACPR en fonction de la puissance d'entrée (16-QAM : débit 50 MHz)

Nous constatons que le résultat d'ACPR du modèle à réponse impulsionnelle s'est quelque peu dégradé pour un débit de 50 MHz, on peut noter une différence d'environ 3dB en saturation par rapport à la co-simulation.

Le tableau ci-dessous présente le temps de simulation ACPR pour les différents modèles implémentés et la co-simulation.

	Co-simulation	Modèle AM/AM AM/PM	Modèle de Volterra un noyau	Modèle de réponse impulsionnelle
Temps de simulation de la chaîne complète	1h. 30 min.	17 sec.	2 min. 38 sec.	2 min. 42 sec.

Tableau III-4 : Temps de simulation de 16-QAM @ 50 MHz

Nous présentons dans la figure II-16 la partie réelle de l'enveloppe temporelle de sortie obtenue par co-simulation et les modèles à réponse impulsionnelle non linéaire et de Volterra à un noyau pour une modulation 16-QAM de débit 50 MHz. Celle ci confirme la dégradation des

performances du modèle à réponse impulsionnelle avec l'augmentation du débit. Le modèle de Volterra un noyau présente toujours la plus mauvaise performance.

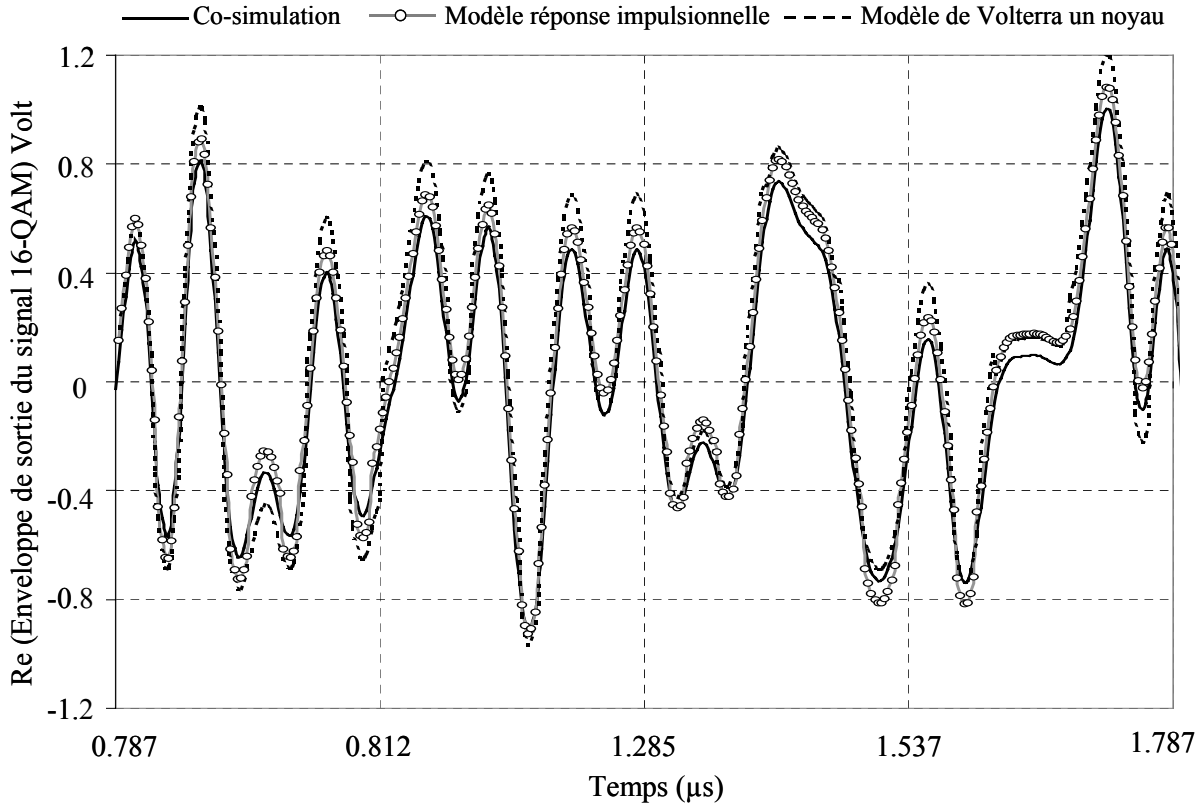


Figure III-16 : Partie réelle de l'enveloppes temporelles d'une 16-QAM @ 50 MHz

Dans la figure suivante (Figure II-17) nous comparons les signaux de sortie des modèles avec la co-simulation, pour un signal à enveloppe constante GMSK de débit 50MHz. On note cette fois-ci que le modèle de Volterra un noyau offre les meilleures performances. Ceci est un résultat attendu car ce modèle a été mis au point pour des applications de la classe des signaux à enveloppe quasi-constante, alors que le modèle à réponse impulsionnelle s'adresse plus particulièrement, dans sa mise au point, à des signaux d'enveloppe variable.

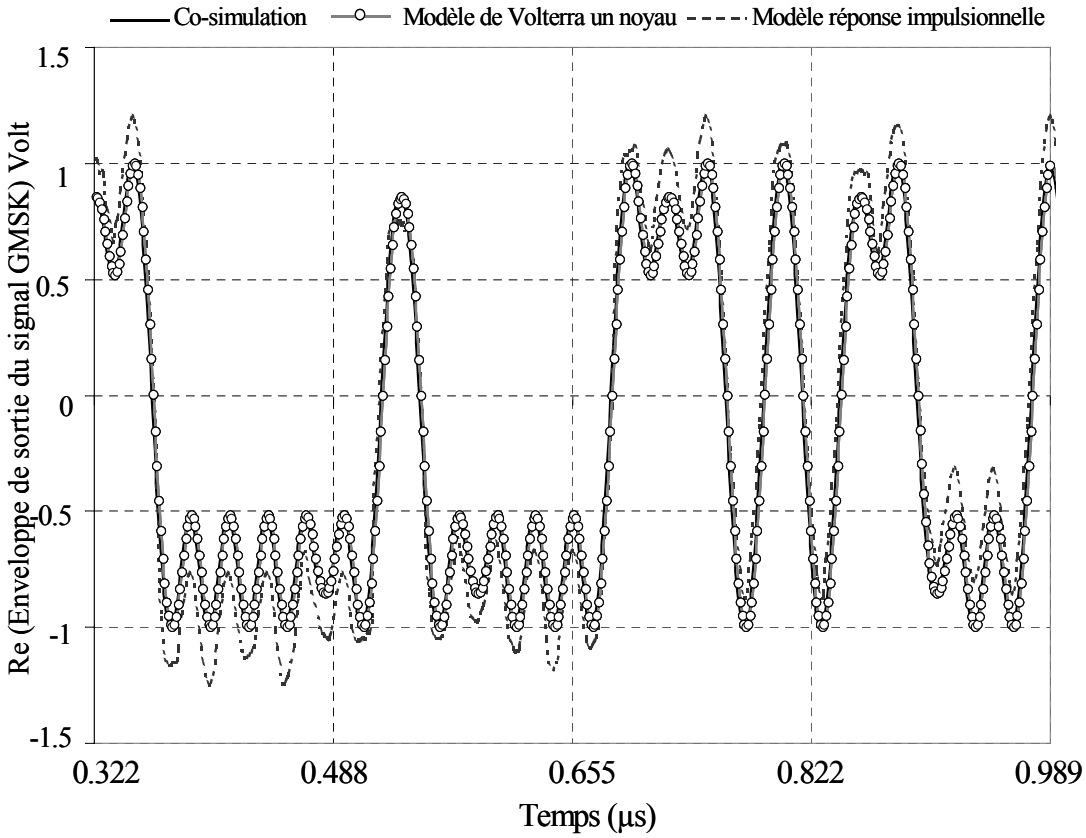


Figure III-17 : Partie réelle de l'enveloppes temporelles d'une GMSK @ 50 MHz

Enfin pour montrer la possibilité de simuler plusieurs blocs circuit en mode co-simulation, nous avons considéré dans le chaîne de communication une simulation de niveau circuit pour le LNA de réception et le mélangeur zero IF (Figure II-18) effectuant la démodulation directe. Les temps de simulation sont encore relativement longs (3 h. 15 min) pour le calcul d'ACPR à la modulation 16-QAM pour un débit de 5 MHz. Cependant avec l'amélioration continue des performances des algorithmes de simulation et des ordinateurs, on peut espérer que ces temps vont rapidement baisser. Notamment lorsque l'on considère la simulation circuit en mode transitoire d'enveloppe rapide (qui ignore les effets de mémoire) dans GoldenGate le temps de simulation de la chaîne n'est plus que de 12 minutes.

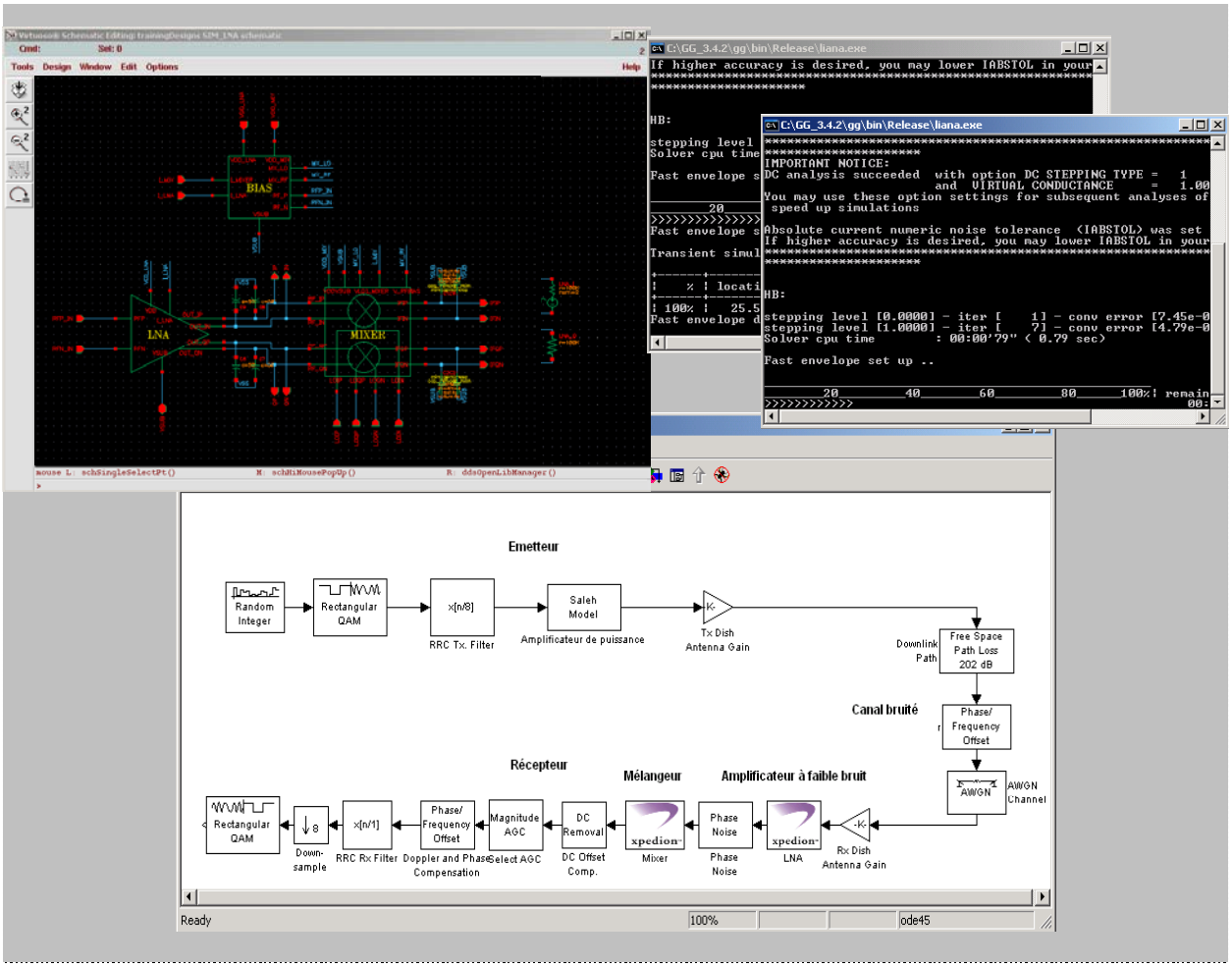


Figure III-18 : La co-simulation entre Simulink et GoldenGate en utilisant un mélangeur

V. Conclusion

Nous avons présenté dans ce chapitre, une interface qui intègre le simulateur circuit GoldenGate™ dans les deux simulateurs système Simulink™ et Scicos™. Cette structure permet l'exécution simultanée des deux simulateurs dans une configuration de co-simulation.

Nous avons commencé ce chapitre en expliquant le principe de co-simulation basé sur une configuration père/fils entre le simulateur système et le simulateur circuit. Pour réaliser cette configuration, nous avons étudié plusieurs mécanismes de communication entre les deux simulateurs, ces mécanismes sont basés sur le mécanisme de communication IPC (Inter Process Communication) sous le système d'exploitation Windows. Nous avons choisi le mécanisme de la mémoire partagée car il est le plus puissant en terme de performances et de parallélisme par rapport aux autres mécanismes.

Ensuite nous avons montré les différents mécanismes de synchronisation entre les deux simulateurs. L'utilisation de ces mécanismes nous permet de bien protéger les données transférées de tentatives d'accès concurrents pour éviter les conflits lors de partages de données entre les deux simulateurs, nous avons choisi les objets évènements.

Finalement nous avons démontré la faisabilité d'une co-simulation circuit-système en mettant en communication les deux simulateurs système Simulink et Scicos avec le simulateur de circuit GoldenGate fonctionnant en transitoire d'enveloppe. Nous avons montré des exemples d'application qui démontrent l'efficacité et l'intérêt de l'approche. Le concepteur de circuit peut ainsi accéder directement aux outils de caractérisation et vérifier précisément les spécifications système. En retour, elle offre au concepteur système (dans un environnement de simulation homogène) une souplesse de modélisation plus importante ; il dispose de tous les modèles comportementaux habituels et peut au besoin accéder à la plus grande précision offerte par la co-simulation. C'est donc un outil qui permet et renforce la communication entre les niveaux de hiérarchie système et circuit.

VI. References

[III-1] **G. Nutt**

« Operating Systems : A Modern Perspective », troisième édition, Department of Computer Science University of Colorado, 2004.

[III-2] **F. Hessel**

« Conception des systèmes hétérogènes multilangages », Thèse de Doctorat de l'Université Joseph Fourier Grenoble, June 2000.

[III-3] **H. Custer**

« Inside WINDOWS NT», Microsoft Press, 1993.

[III-4] **C. Blaess**

« Programmation système en C sous Linux », deuxième édition, 2002.

[III-5] **P. Villani**

« Programming Win32 Under the Api », CMP Books, August 2001.

[III-6] **M. Brain, R. Reeves**

« Win 32 System Services: The Heart of Windows 98 and Windows 2000 », troisième édition, Prentice Hall PTR, December 2000.

[III-7] **N. Grattan, M. Brain**

« Windows CE 3.0 Application Programming », Prentice Hall PTR, October 2000.

[III-8] **A. Bennadji, A. Soury, E. Ngoya and R. Quéré**

« Implementation of behavioral models in system simulator and RF circuit/system co-simulation », INMMC Workshop, Aveiro, Portugal, Janvier 2006.

[III-9] **M. Charles, A. Terrice**

« Create Communications Programs for Windows 95 with the Win32 Comm API », Microsoft Systems Journal, December 1994.

[III-10] **G. Nebbett**

« Windows NT/2000 Native API Reference », Mtp Mcmillan Technical Publishing, 304 pages, February 2000.

Conclusion Générale

Conclusion Générale

Le travail de cette thèse est décomposé en deux parties, la première partie a été consacrée à l'implémentation des modèles comportementaux dans des environnements de simulation système tels que Scicos et Simulink, la deuxième partie a traité la co-simulation circuit-système avec une application pratique, sur la base des simulateurs systèmes Scicos et Simulink et du simulateur circuit GodenGate.

Nous avons développé dans la première partie de ce travail de thèse deux approches mathématiques pour implémenter les modèles d'amplificateur dans les deux simulateurs Scicos et Simulink, l'idée de base consiste à reformuler les équations des modèles de Volterra de façon à améliorer le temps de calcul au niveau système. Ces deux approches permettent d'assurer une prédiction des réponses précise, rapide et stable afin d'offrir des outils de simulation « système » robustes, précis et qui soient de connivence avec les outils « circuit ».

La première méthode que nous avons adopté est basée sur des caractéristiques fréquentielles par fonction de Padé. Cette méthode permet d'éliminer l'intégrale de convolution, pour ne pas pénaliser le temps de calcul et pour utiliser une implémentation de modèle sous la forme d'un filtre à réponse impulsionnelle finie (FIR) ou infinie (IIR), est d'exprimer les réponses des dispositifs sous la forme d'équations différentielles.

La deuxième méthode permet aussi d'éliminer l'intégrale de convolution en utilisant la décomposition des noyaux de Volterra ou la réponse impulsionnelle en série de fonctions séparables. Cette décomposition permet de dissocier les deux variables de la fonction permettant une utilisation plus souple du formalisme.

Dans la deuxième partie de ce travail, nous avons développé un prototype de co-simulation circuit-système qui permet la co-simulation entre le simulateur circuit GoldenGate développé en collaboration entre Xlim et la société *Xpedion Design System* et les deux simulateurs système Scicos et Simulink. Ce prototype va nous permettre de simuler des chaînes de communication complètes en utilisant des modèles d'amplificateur ou mélangeur du simulateur de niveau transistor, afin d'obtenir la précision la plus grande possible. Le principe de fonctionnement de ce prototype est basé sur une configuration père/fils entre le simulateur système et le simulateur circuit. La communication entre les deux simulateurs est réalisée par le mécanisme de communication IPC (Inter Process Communication) sous le système d'exploitation Windows.

Nous avons montré par des exemples d'application pratiques l'efficacité des modèles de Volterra dans le contexte critique en temps de calcul de la simulation système De même nous avons démontré la faisabilité et l'intérêt de la co-simulation circuit-système. Les temps de simulation restent encore important dans la co-simulation, mais sont dans un ordre de grandeur tout à fait satisfaisant pour la validation de segments de système. Il est tout à fait évident que ce genre d'outil devrait améliorer la communication entre les mondes circuit et système.

Dans ce travail nous nous sommes appesantis principalement sur les modèles unilatéraux qui ne prennent pas en compte les désadaptations d'impédance. La raison principale en est que les simulateurs systèmes que nous avons considéré et tous les simulateurs systèmes généraux travaillant sur le mode data flow, ne permettent pas de gérer la dualité courant/tension qui seule permet de prendre en compte la désadaptation d'impédance. La suite logique de ce travail serait donc d'étudier l'implémentation optimale dans les simulateurs système des modèles avec désadaptation d'impédance. On peut penser notamment à l'extension du modèle de Volterra dit de « paramètres S non linéaires » récemment développé à Xlim par A. Soury. Les extensions du simulateur Scicos vers la résolution des systèmes d'équation implicites devraient permettre une implémentation efficace de ce type de modèle.

ANNEXES

A1. Calcul des coefficients de noyaux de Volterra

Considérons l'expression de l'approximation du noyau de Volterra :

$$H\left(\left|\tilde{X}(t)\right|,\Omega\right)=\frac{\sum_{n=0}^{N}\alpha_n\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^n}{1+\sum_{m=1}^{M}\beta_m\left(\left|\tilde{X}(t)\right|\right).(j.\Omega)^m} \quad \text{avec} \quad N \le M \tag{A1-1}$$

Les deux noyaux de Volterra $H_1\left(\left|\tilde{X}(t)\right|\Omega\right)$ et $H_2\left(\left|\tilde{X}(t)\right|\Omega\right)$ sont connus (données d'extraction) et Ω est l'écart avec la fréquence dite de référence (*fréquence centrale*).

Pour calculer les coefficients $\alpha_n\left(\left|\tilde{X}(t)\right|\right)$ et $\beta_m\left(\left|\tilde{X}(t)\right|\right)$, il faut écrire l'équation (A1-1) pour Q valeurs de Ω, sous forme matricielle comme indiqué ci-dessous :

$$[H]=[X].[a] \tag{A1-2}$$

où

$$[H]=\left[H_{\Omega 1},H_{\Omega 2},.....,H_{\Omega N}\right]^T,\ [a]=\left[\alpha_0,\alpha_1,.....,\alpha_N,\beta_1,\beta_2,.....,\beta_M\right]^T$$

$$[X]=\begin{bmatrix} 1 & j\Omega_1 & . & . & (j\Omega_1)^N & -H\left(\left|\tilde{X}(t)\right|\Omega_1\right).j\Omega_1 & . & . & . & -H\left(\left|\tilde{X}(t)\right|\Omega_1\right).(j\Omega_1)^M \\ 1 & j\Omega_2 & & & & & & & & \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 1 & j\Omega_Q & & & (j\Omega_Q)^N & -H\left(\left|\tilde{X}(t)\right|\Omega_Q\right).j\Omega_Q & & & & -H\left(\left|\tilde{X}(t)\right|\Omega_Q\right).(j\Omega_Q)^M \end{bmatrix}$$

En général $Q > N+M$, donc la résolution de ce système d'équation pour calculer les coefficients α_n et β_m nécessite l'utilisation de la méthode de moindre carré [A1-1] pour chaque niveau de puissance d'entrée $\left|\tilde{X}(t)\right|$, l'expression de cette méthode peut s'écrire sous la forme :

$$[a]=\left\lfloor[X]^T.[X]\right\rfloor^{-1}.[X]^T.[H] \tag{A1-3}$$

Références

[A1-1] **M. Schatzman**

« Analyse Numérique», Inter Editions, 1991.

A2. Méthode d'identification des pôles-zéros

Comme nous avons mentionné dans le chapitre II, l'approximation de Padé peut présenter dans temps à autre des problèmes de stabilité. Pour éviter ce problème nous utilisons la technique d'identification des pôles-zéros [A2-1].

Un système linéaire peut être représenté par sa fonction de transfert $H(\omega)$, dans le domaine temporal il est représenté par la réponse impulsionnelle $h(t)$.

$$H(\omega) \underset{FT^{-1}}{\overset{FT}{\rightleftarrows}} h(t) \tag{A2-1}$$

Comme les signaux, un système passe-bande peut être représenté comme un système passe-bas équivalent. Dans ce cas, nous pouvons exprimer la fonction de transfert $H(\omega)$ comme suit :

$$H(\omega) = \tilde{H}(\omega - \omega_C) + \tilde{H}^*(-\omega - \omega_C) \tag{A2-2}$$

Toutes les fonctions transfert peuvent être exprimées sous la forme d'une fonction raisonnable (approximation de Padé) où $N \le M$ et $N' \le M'$.

$$H(\omega) = \frac{P(\omega)}{Q(\omega)} = \frac{\sum_{n=0}^{N} a_n.(j.\omega)^n}{1 + \sum_{m=1}^{M} b_n.(j.\omega)^m} \quad \text{ou} \quad \tilde{H}(\omega) = \frac{\tilde{P}(\omega)}{\tilde{Q}(\omega)} = \frac{\sum_{n=0}^{N'} \tilde{a}_n.(j.\omega)^n}{1 + \sum_{m=10}^{M'} \tilde{b}_n.(j.\omega)^m}$$

Les fonctions de transfert $H(\omega)$ et $\tilde{H}(\omega)$ sont respectivement caractérisées par les coefficients a_n, b_m et $\tilde{a}_n$, $\tilde{b}_m$. En factorisant ces fonctions rationnelles, nous obtenons la représentation de pôles-zéros suivante :

$$H(\omega) = \frac{P(\omega)}{Q(\omega)} = \mu \frac{\prod_{n=1}^{N} (j.\omega - z_n)}{\prod_{m=1}^{M} (j.\omega - p_m)} \quad \text{ou} \quad \tilde{H}(\omega) = \frac{\tilde{P}(\omega)}{\tilde{Q}(\omega)} = \tilde{\mu} \frac{\prod_{n=1}^{N'} (j.\omega - \tilde{z}_n)}{\prod_{m=1}^{M'} (j.\omega - \tilde{p}_m)}$$

L'identification de la fonction de transfert consiste à trouver les coefficients de l'approximation de Padé ou les pôles (p_m ou $\tilde{p}_m$) et les zéros (z_n or $\tilde{z}_n$). Pour garantir la stabilité de la fonction, tous les pôles doivent avoir une partie réelle strictement négative. Pour identifier les pôles et les zéros, nous devons premièrement approximer la fonction par l'approximation de Padé et deuxièmement trouver les racines du numérateur et du dénominateur.

Pour assurer la stabilité de la fonction, nous utilisons une décomposition en fréquence carrée ω^2. La partie réelle et imaginaire de $H(\omega)$ peut être exprimée sous la forme suivante :

$$\Re e\{H(\omega)\} = \frac{\Re e\{P(\omega).Q^*(\omega)\}}{Q(\omega).Q^*(\omega)} = \frac{c_0 + c_1.\omega^2 + c_2.\omega^4 + \cdots + c_K.\omega^K}{1 + \beta_1.\omega^2 + \beta_2.\omega^4 + \cdots + \beta_M.\omega^{2.M}} \qquad \text{(A2- 3)}$$

$$\Im m\{H(\omega)\} = \frac{\Im m\{P(\omega).Q^*(\omega)\}}{Q(\omega).Q^*(\omega)} = \frac{d_1.(j.\omega) + d_2.(j.\omega)^3 + \cdots + d_K.(j.\omega)^L}{1 + \beta_1.\omega^2 + \beta_2.\omega^4 + \cdots + \beta_M.\omega^{2.M}} \qquad \text{(A2- 4)}$$

où

$$\begin{cases} K = N + M \quad et \quad L = N + M - 1 \quad si \ (N+M) \ est\ pair \\ K = N + M - 1 \quad et \quad L = N + M \quad si \ (N+M) \ est\ impair \end{cases} \qquad \text{(A2- 5)}$$

Les coefficients c_i, d_i et β_i sont purement réels. Pour résoudre ces équations on utilise la technique de résolution par le moindre carré.

$$\left[\begin{array}{cccc|ccc|ccc} 1 & \omega_1^2 & \cdots & \omega_1^K & 0 & \cdots & 0 & -\omega_1^2.\Re e\{H(\omega_1)\} & \cdots & -\omega_1^{2.M}.\Re e\{H(\omega_1)\} \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 1 & \omega_Z^2 & \cdots & \omega_Z^K & 0 & \cdots & 0 & -\omega_Z^2.\Re e\{H(\omega_Z)\} & \cdots & -\omega_Z^{2.M}.\Re e\{H(\omega_Z)\} \\ \hline 0 & \cdots & \cdots & 0 & (j.\omega_1) & \cdots & (j.\omega_1)^L & -\omega_1^2.\Im m\{H(\omega_1)\} & \cdots & -\omega_1^{2.M}.\Im m\{H(\omega_1)\} \\ \vdots & & & & \vdots & \ddots & \vdots & \vdots & & \vdots \\ \vdots & & & & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & \cdots & 0 & (j.\omega_Z) & \cdots & (j.\omega_Z)^L & -\omega_Z^2.\Im m\{H(\omega_Z)\} & \cdots & -\omega_Z^{2.M}.\Im m\{H(\omega_Z)\} \end{array}\right] \times \left[\begin{array}{c} c_0 \\ \vdots \\ \vdots \\ c_K \\ \hline d_1 \\ \vdots \\ \vdots \\ d_L \\ \hline \beta_1 \\ \vdots \\ \vdots \\ \beta_M \end{array}\right] = \left[\begin{array}{c} \Re e\{H(\omega_1)\} \\ \Re e\{H(\omega_2)\} \\ \vdots \\ \vdots \\ \Re e\{H(\omega_Z)\} \\ \hline \Im m\{H(\omega_1)\} \\ \Im m\{H(\omega_2)\} \\ \vdots \\ \vdots \\ \Im m\{H(\omega_Z)\} \end{array}\right]$$

Après l'utilisation de la méthode de *Eigenvalue* [A2-2] pour trouver les racines de dénominateur, nous obtenons M pôles notés p_i. La relation entre les pôles recherchés et les pôles désirés est la suivante:

$$\boxed{p_i = \pm\sqrt{\hat{p}_i}} \qquad \text{(A2- 6)}$$

Après, nous utilisons seulement les pôles stables ($\Re e\{p_i\} < 0$), les pôles instables sont enlevés dans la majorité des cas.

Références

[A2-1] A. Soury

« Identification of Transfer Functions and Impulse Response », Rapport Interne Xpedion, April 2004.

[A2-2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst

« Templates for the Solution of Algebraic Eigenvalue Problems », SIAM, Philadelphia, 2000.

Communications dans les congres internationaux

[1] **A. Bennadji, A. Soury and E. Ngoya**
« Implementation of dynamic Volterra model in system Environment », International Workshop on Electronics and System Analysis, Bilbao Espagne, Octobre 2004.

[2] **A. Bennadji, A. Soury, S.Mons, E. Ngoya, R.Quéré**
« Implementation of dynamic Volterra model in Scicos Environment », 1st Scilab International Conference, Paris, Janvier 2004.

[3] **A. Layec, A. Bennadji, J.C. Nallatamby, R. Quéré and E. Ngoya**
« Modelling of a Communication Chain Components and Principles of Simulation in the Scilab/Scicos environment », IEEE Conference on Control Applications, CCA05 Toronto, Août 2005.

[4] **A. Bennadji, A. Layec, A. Soury, A. Mallet, E. Ngoya and R. Quéré**
« Modeling of a communication chain with implementation of a Volterra power amplifier model for efficient system level simulation », 8th European Conference on Wireless Technology - European Microwave Week 2005, Paris, Octobre 2005.

[5] **A. Bennadji, A. Soury, E. Ngoya and R. Quéré**
« Implementation of behavioural models in system simulator and RF circuit/system co-simulation», Integrated Nonlinear Microwave and Millimeter-wave Circuits (INMMC) Workshop, Aveiro, Portugal, Janvier 2006.

Communications dans les congres nationaux

[6] **A. Bennadji, A. Soury, E. Ngoya, R. Quéré**
« Implémentation des modèles comportementaux dans des environnements de simulation système », 14eme Journées Nationales Micro-ondes (JNM),5A-3, Nantes, Mai 2005.

RESUME

Ce travail est décomposé en deux parties, la première partie concerne l'implémentation de modèles comportementaux d'amplificateur de puissance dans des environnements de simulation système tels que Simulink et Scicos, nous avons développé dans cette partie deux approches mathématiques pour implémenter les modèles d'amplificateur dans les deux simulateurs, l'idée de base consiste à reformuler les équations des modèles de Volterra de façon à améliorer le temps de calcul au niveau système. Ces deux approches permettent d'assurer une prédiction des réponses précise, rapide et stable afin d'offrir une simulation système robuste. Dans la deuxième partie de ce travail, nous avons développé un prototype de co-simulation circuit–système sur la base des simulateurs systèmes Scicos et Simulink et du simulateur circuit GoldenGate. Ce prototype est basé sur le mécanisme de communication entre processus (IPC) sous le système d'exploitation Windows. Ce prototype renforce la communication entre les niveaux de hiérarchie système et circuit et offre une grande précision.

ABSTRACT

This work is organized in two parts, the first part concerns the implementation of behavioral models of power amplifier in system level simulators such as Simulink and Scicos. We developed in this part two mathematical approaches to implement the amplifier models in the two simulators, the basic idea consists in reformulating the equations of Volterra models in order to improve the computing time on the system level. These two approaches make it possible to ensure simulation speed and stability in order to offer a robust system simulation. In the second part of this work, we developed a prototype of circuit-system co-simulation on the basis of the systems simulators Scicos and Simulink and the GoldenGate circuit simulator. This prototype is based on the Inter-Process Communication mechanism (IPC) under Windows operating system. This prototype reinforces the communication between the system hierarchy levels and circuit and offers a high degree of accuracy.

MOTS CLES

Implémentation de modèles comportementaux

Amplificateur de puissance

Simulation système

Co-simulation circuit système

Communication entre processus (IPC)