TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

BPC	Basic Predistortion Cell
CW	Continuous Wave
BRAM	Block Random Access Memory
COM	Component Object Model
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
IIR	Infinite Impulse Response
LMS	Least Mean Squares
LS	Least Squares
NARMA	Nonlinear Autoregressive Moving Average
QAM	Quadrature Amplitude Modulation
RF	Radiofréquence
RLS	Recursive Least Squares
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VSA	Vector Signal Analyzer
VSG	Vector Signal Generator
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

LISTE DES SYMBOLES ET UNITÉS DE MESURE

A	Ampère
Gbps	Gigabits par seconde
Mbps	Mégabits par seconde
MSPS	Mega Samples Per Second
V	Volt
W	Watt

CHAPITRE 1

INTRODUCTION

1.1 Aperçu général du projet

Au cours des dernières décennies, nous avons assisté à l'explosion de la quantité d'appareils sans-fil en circulation. Leur autonomie est une des caractéristiques les plus importantes recherchées par l'industrie. En ce sens, on comprend que des efforts colossaux sont déployés afin d'améliorer l'efficacité énergétique de ces appareils. Au-delà des préoccupations strictement utilitaires, l'efficacité énergétique est aussi très importante d'un point de vue environnemental. En effet, elle permet de réduire la consommation énergétique, mais aussi, de réduire la quantité de déchets causés par les piles. En raison du nombre important d'appareils en circulation, une amélioration à petite échelle permet d'avoir un impact écologique considérable. Afin de maximiser l'efficacité des efforts de recherche, on vise davantage les composantes responsables de la plus grande consommation d'énergie.

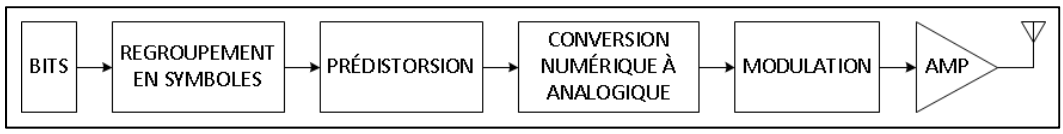


Figure 1.1 Chaîne de transmission typique

À la Figure 1.1, on retrouve un schéma bloc de la chaîne de transmission typique qu'on peut retrouver dans ces appareils. Dans un premier temps, les bits à transmettre sont regroupés en symboles. Chaque symbole représente deux niveaux de tension, soit les parties réelle et imaginaire d'un nombre complexe. Un système de traitement numérique de signal permet d'effectuer une prédistorsion des signaux afin de compenser certaines caractéristiques indésirables du reste de la chaîne de transmission. Ensuite, le signal numérique est converti en niveaux de tension analogiques. Ceux-ci sont utilisés pour moduler un signal RF. Ce signal est ensuite acheminé à un amplificateur de puissance qui permet d'augmenter son intensité avant d'être transmis par l'antenne.

L'amplificateur de puissance RF présent dans ces appareils traite donc une puissance électrique considérable: il doit être le plus efficace possible. Comme nous le verrons, les amplificateurs de puissance ont des caractéristiques intrinsèques qui les rendent plus difficiles à utiliser. Leur réponse n'est pas toujours linéaire et ils montrent souvent des effets mémoire. Ces problèmes rendent leur sortie difficile à prévoir en fonction de leur entrée.

Dans le but d'augmenter l'efficacité énergétique des amplificateurs de puissance RF, des mécanismes de reconfiguration ont été développés. Ceux-ci ont souvent pour effet d'exacerber les problèmes ci-haut mentionnés. Les méthodes de prédistorsion numériques, telles que celles basées sur une structure NARMA, se posent en solution à ces problèmes.

Dans ce projet, nous voyons le cas d'un amplificateur de puissance reconfigurable dont la réponse entrée/sortie présente une courbe non monotone. Cette caractéristique, peu étudiée jusqu'à présent, est un problème lorsqu'on souhaite appliquer une prédistorsion de type NARMA, qui permet normalement d'améliorer efficacement la linéarité des amplificateurs RF, avec ou sans effets mémoire. Au meilleur de nos connaissances, il n'y a pas de façon simple ou efficace d'obtenir un module de prédistorsion NARMA dans un contexte de non monotonie.

Dans ce travail, nous verrons dans un premier temps des notions théoriques relatives aux amplificateurs de puissance RF ainsi que certaines méthodes de prédistorsion existantes. Par la suite, nous étudierons le cas d'un amplificateur de puissance RF reconfigurable. L'inefficacité des techniques actuelles à réaliser une prédistorsion de type NARMA sera expliquée. Ensuite, une solution sera proposée et implémentée en simulation. Finalement, nous verrons un système de prédistorsion conçu et partiellement implémenté qui permettra de réaliser une prédistorsion de type NARMA. Il est conçu de façon à offrir une flexibilité et une facilité de configuration qui permettront d'augmenter l'efficacité des chercheurs qui l'utilisent.

1.2 Cadre de la recherche

Dans ce travail de recherche, le focus est d'abord mis sur la prédistorsion numérique d'amplificateurs RF, dans le cas où une reconfiguration dynamique des transistors crée une réponse statique en tension non monotone. Bien que différentes techniques de prédistorsion aient été étudiées, ce travail ne se veut pas une comparaison exhaustive de l'ensemble des techniques qui existent. À ce sujet, une comparaison intéressante est faite dans (Gilabert, 2008). La technique de prédistorsion privilégiée est celle de type NARMA, notamment en raison de sa facilité d'utilisation dans un contexte de laboratoire. Ce travail ne s'attarde pas à la question de stabilité ni à celle de l'optimisation des paramètres, tels que l'ordre du filtre, l'ordre des polynômes ou la valeur des délais. Ces sujets ont déjà été traités dans la littérature, par exemple dans (Gilabert *et al.*, 2008).

1.3 Contributions à la recherche

Ce projet permet d'effectuer le premier traitement de la réponse non monotone d'un amplificateur sous reconfiguration hardware. Une expression mathématique de forme explicite a été trouvée. Elle permet de calculer directement les valeurs d'un module de prédistorsion de type NARMA à partir du modèle de l'amplificateur obtenu au terme d'une séquence d'entraînement. Cela ouvre une nouvelle voie dans la conception de systèmes fortement non linéaires en assurant la possibilité d'utiliser une prédistorsion de type NARMA pour compenser les non linéarités, même en cas de non monotonie. Finalement, une plateforme de prédistorsion flexible et configurable a été conçue et partiellement réalisée. Celle-ci offrira aux chercheurs l'opportunité d'accélérer leurs travaux et de valider leurs résultats de façon expérimentale.

CHAPITRE 2

THÉORIE SUR LES AMPLIFICATEURS DE PUISSANCE RF

2.1 Introduction

Dans ce chapitre, nous étudierons les bases théoriques qui permettent de comprendre certaines des caractéristiques les plus importantes de l'amplificateur de puissance RF, dans le contexte de transmission d'un signal modulé. Les imperfections principales des amplificateurs de puissance seront abordées et des techniques de linéarisation logicielles pouvant aider à pallier à ces problèmes seront ensuite vues. Finalement, l'applicabilité de deux de ces méthodes sera analysée dans le cas d'un milieu de développement.

Avant d'aller plus loin, faisons un tour d'horizon rapide du cas d'utilisation d'un amplificateur de puissance RF, qui sera celui considéré dans ce mémoire. Un système de transmission RF moderne est constitué de plusieurs composantes fondamentales. Dans un premier temps, un système numérique calcule une séquence de bits à transmettre. Les bits sont ensuite regroupés en symboles. Ces symboles sont associés à un niveau de tension électrique. Ces tensions servent ensuite à moduler une onde porteuse à fréquence RF. Dans le cas d'une modulation en quadrature (ex : 16-QAM), il s'agit de l'addition de deux ondes sinusoïdales de même fréquence avec une phase de 90 degrés. L'onde résultante est ensuite acheminée à l'amplificateur de puissance RF. Le rôle de l'amplificateur est d'augmenter la puissance du signal au niveau désiré, avant que ce dernier n'atteigne l'antenne pour être transmis.

2.1.1 Linéarité

La linéarité d'un amplificateur de puissance RF est une de ses caractéristiques les plus importantes. Dans un amplificateur idéal, la puissance en sortie varie proportionnellement à la puissance d'entrée. Autrement dit, le gain est constant. Tel que vu dans (Cripps, 2006), pour un amplificateur réel typique, on considère que sa courbe est linéaire jusqu'au point

P_{1dB}, où il atteint une compression de 1 dB. À partir de ce point, le gain chute rapidement : on dit alors que l'amplificateur entre en compression. La figure ci-dessous illustre une courbe de compression typique. Au point P_{1dB}, le gain a chuté de 1 dB par rapport à la droite représentant la réponse idéale linéaire de l'amplificateur.

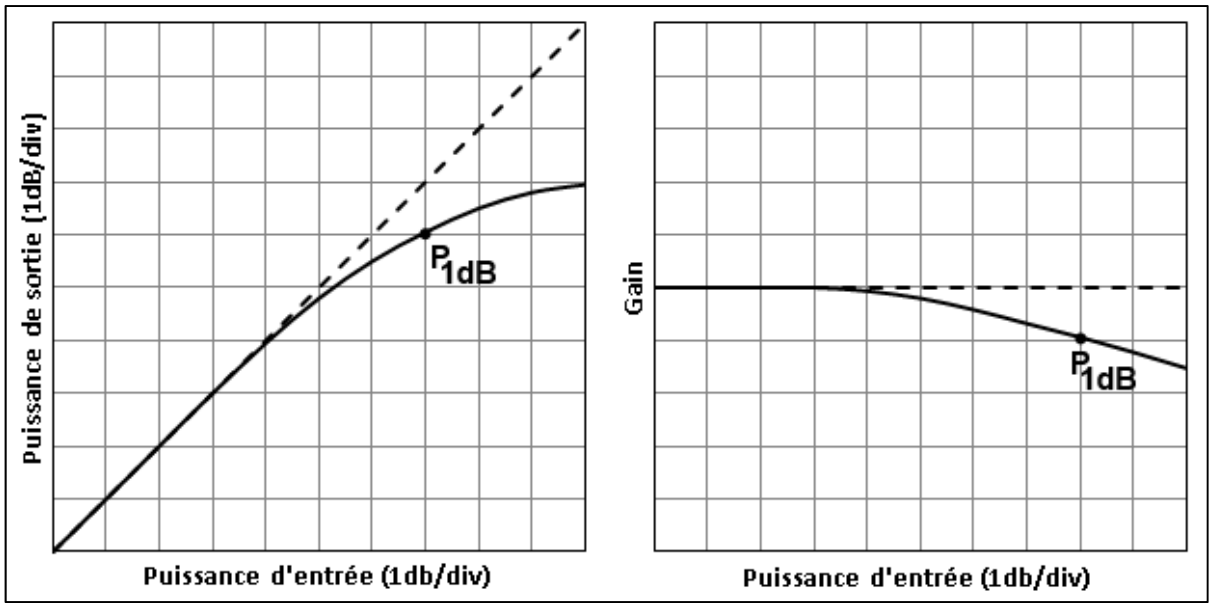


Figure 2.1 Courbes typiques de la puissance de sortie en fonction de la puissance d'entrée et du gain d'un amplificateur de puissance RF

La partie linéaire de sa réponse en puissance est celle qui est la plus facile à utiliser d'un point de vue pratique. Néanmoins, se limiter à la partie linéaire a un impact sur l'efficacité de l'amplificateur, définie par le ratio entre la puissance de la fondamentale du signal RF de sortie et la puissance moyenne prise de la batterie qui est nécessaire pour alimenter l'amplificateur. À polarisation fixe des transistors RF de l'amplificateur, lorsqu'on réduit la puissance de l'entrée (et donc aussi la puissance de sortie) afin de rester dans la partie linéaire, le courant moyen pris de la batterie reste inchangé, et par conséquent on abaisse l'efficacité énergétique de l'amplificateur. Pour contrer cette baisse de l'efficacité lorsque la puissance RF est réduite, une méthode employée est de bloquer le fonctionnement d'une partie du circuit amplificateur (généralement en supprimant la polarisation DC de certains transistors RF dans cette partie de l'amplificateur) afin de réduire la puissance moyenne prise

de la batterie. Dans le contexte d'un appareil portable, l'efficacité énergétique joue un rôle crucial. Du point de vue des constructeurs, elle est synonyme d'une meilleure autonomie, ce qui constitue évidemment un argument de vente important. D'un point de vue écologique, elle permet de réduire la demande énergétique et prolonge la durée de vie des batteries.

2.1.2 Effet mémoire

L'effet mémoire est un sujet complexe largement étudié. Il traite de l'impact temporel des entrées et sorties de l'amplificateur sur sa réponse. On considère généralement deux types d'effets mémoire, soit électrique et thermique. La citation suivante résume bien ces derniers:

> Electrical memory effects are caused by frequency-dependent impedances due to energy-storage elements (capacitors and inductors), which introduce phase shifts in the electric signals (and the associated time dependence in the PA response). Thermal memory effects are caused by time-dependent variations in junction temperatures of semiconductor devices, function of the intensity of the modulated electric signals being amplified, thereby introducing time constants in the electrical transfer functions of the devices. (Constantin *et al.*, 2012)

Les effets mémoire peuvent donc se manifester avec plus ou moins d'intensité en fonction de multiples paramètres, dont la puissance du signal. Pour des signaux de faible puissance, les effets mémoire tendent à être moins importants qu'à haute puissance. C'est d'ailleurs pourquoi on retrouve un nombre considérable de papiers qui traitent des effets mémoire dans le cas des stations de base. On note toutefois un intérêt plus important envers les appareils portables ces dernières années, notamment en raison de l'importance cruciale que devient l'efficacité énergétique et de la multiplication des appareils portables.

Les systèmes de prédistorsion numériques sont limités quant à l'impact qu'ils peuvent avoir sur les effets mémoire en fonction de leur fréquence d'opération. Ainsi, des effets mémoire dont le terme est trop court seront généralement traités avec des techniques de prédistorsion

matérielles. Pour de plus amples détails sur les causes des effets mémoires, la lecture de (Vuolevi, 2003) est fortement recommandée.

2.1.3 Amélioration de l'efficacité énergétique par reconfiguration hardware

L'objectif de la reconfiguration hardware est de permettre à l'amplificateur de s'adapter à la puissance du signal à transmettre de façon à réduire au maximum la puissance moyenne consommée de la batterie. Il existe plusieurs techniques de reconfiguration hardware. On retrouve par exemple l'ajustement de la polarisation de l'amplificateur en fonction de la puissance d'entrée. Cet ajustement peut même aller jusqu'à la désactivation complète de cellules de transistors lorsque la puissance diminue. Leur réactivation peut être faite lorsque la puissance augmente de nouveau.

Comme on peut s'en attendre, une reconfiguration hardware peut introduire des non-linéarités importantes dans la réponse entrée/sortie de l'amplificateur. Les techniques de prédistorsion présentées aux sections suivantes permettent de contrecarrer ces effets indésirables.

2.2 Techniques de linéarisation logicielles

Que ce soit pour compenser les effets non linéaires intrinsèques d'un amplificateur ou pour compenser les effets collatéraux de mécanismes de reconfiguration hardware qui provoquent la non linéarité, il existe une multitude de techniques de prédistorsion qui permettent d'effectuer une linéarisation. Dans cette section, nous considérerons deux techniques principales qui ont été considérées dans le cadre de ce travail.

2.2.1 Rétroaction cartésienne

Dans (SungWon, Holloway et Dawson, 2008), les auteurs proposent une méthode de prédistorsion : la rétroaction cartésienne. Le principe consiste à récupérer une fraction du signal de sortie de l'amplificateur. Ce signal est ensuite comparé à l'entrée de l'amplificateur.

La différence entre ces valeurs permet d'ajuster une table de correspondance qui génère le signal d'entrée de l'amplificateur. Lorsqu'une certaine stabilité est atteinte, le circuit d'adaptation est coupé, ne laissant que la table de correspondance en fonction. Advenant un changement des conditions d'opération, l'adaptation peut être réactivée jusqu'à ce qu'un nouvel état d'équilibre soit trouvé.

La simplicité de cette technique fait en sorte de réduire au maximum l'impact de la méthode de prédistorsion sur l'efficacité énergétique. Néanmoins, son plus grand désavantage est qu'elle nécessite que l'amplificateur ne présente pas d'effets mémoires très importants.

2.2.2 Prédistorsion NARMA (v1)

Dans (Montoro *et al.*, 2007), les auteurs présentent une méthode de prédistorsion en trois étapes :

1. Génération d'une série de données entrée/sortie de l'amplificateur;
2. Calcul du modèle NARMA de l'amplificateur;
3. Calcul du modèle de prédistorsion par inversion du modèle de l'amplificateur.

Advenant le cas où les conditions d'opération de l'amplificateur viennent à varier, on peut exécuter les étapes à nouveau afin de générer un modèle de prédistorsion valide dans les nouvelles conditions.

Le modèle NARMA consiste en une combinaison de filtres FIR et IIR complexes disposés en cascade, dont les coefficients ont été remplacés par des fonctions non linéaires. À la Figure 2.2, on voit à droite, le modèle de l'amplificateur sous la forme d'une structure NARMA, et à gauche, le module de prédistorsion (aussi sous la forme d'une structure NARMA légèrement modifiée) qui permet de linéariser l'amplificateur. Les blocs FIR et IIR dont les entrées sont décalées ont pour but de permettre de modéliser et compenser les effets mémoire de l'amplificateur.

Notons que les quantités $x(k)$ et $y_f(k)$ de la Figure 2.2 sont les tensions réelles qui modulent les signaux RF à l'entrée et à la sortie de l'amplificateur. En revanche, la quantité $y_0(k)$ et les valeurs intermédiaires à l'intérieur du circuit de prédistorsion sont des valeurs numériques qui permettent de calculer les tensions de $x(k)$ que le convertisseur numérique/analogique doit générer. Elles ne sont donc pas représentatives de l'intensité des signaux numériques à l'intérieur du circuit réel de prédistorsion.

À la Figure 2.2, les blocs de la colonne de gauche du modèle de l'amplificateur constituent le filtre FIR, tandis que ceux de droite du même modèle constituent le filtre IIR. Les boîtes qui contiennent l'indication Z^{-1} sont des délais d'un cycle machine. La valeur qui arrive à leur entrée est donc stockée et transmise à la sortie au cycle suivant seulement. Les blocs ayant l'indication $\hat{F}_i$ et $\hat{G}_i$ représentent des fonctions polynomiales qui prennent un scalaire en entrée et retournent un nombre complexe en sortie. Finalement, les boîtes contenant l'indication $|Z|$ retournent le module du nombre complexe à leur entrée.

Une fois l'amplificateur modélisé et linéarisé par le module de prédistorsion, nous obtenons l'égalité $y_0(k) = y_f(k)$, ce qui traduit une réponse d'amplificateur prédictible. L'entrée du système $v(k)$ est donc le signal que l'on souhaite amplifier par un gain linéaire. En gris foncé, on retrouve l'implémentation des différentes fonctions f_i et g_j, qui consistent à multiplier le nombre complexe en entrée par un autre nombre complexe dont la valeur dépend de l'amplitude de l'entrée. Chaque fonction est représentée par un bloc constitué de ces éléments. On nomme donc ce bloc BPC pour Basic Predistortion Cell. Notons que le BPC f_0 contient les réponses statiques AM-AM (amplitude-amplitude) et AM-PM (amplitude-phase) de l'amplificateur, c'est-à-dire mesurées dans des conditions qui excluent les effets mémoire de l'amplificateur. Cela peut être réalisé en appliquant un signal sinusoïdal d'amplitude et de phase constantes à l'entrée de l'amplificateur, et en mesurant les conversions AM-AM et AM-PM de l'entrée à la sortie en régime permanent. Ensuite, on change l'amplitude à l'entrée et on reprend la mesure des conversions AM-AM et AM-PM en régime permanent, et ainsi de suite jusqu'à ce qu'on couvre la plage d'amplitude désirée pour les courbes statiques.

Pour récapituler, les fonctions

$$f_i\big(x(k-\tau_i)\big) = x(k-\tau_i)\cdot \hat{F}_i(|x(k-\tau_i)|) \tag{2.1}$$

et

$$g_j\left(y\big(k-\tau_j\big)\right) = y\big(k-\tau_j\big)\cdot \hat{G}_j\big(\big|y\big(k-\tau_j\big)\big|\big) \tag{2.2}$$

sont identiques dans le module de prédistorsion et dans le modèle de l'amplificateur pour $i, j > 0$.

Il a précédemment été mentionné que les fonctions $\hat{F}_i$ et $\hat{G}_j$ sont représentées par des fonctions polynomiales dans le modèle de l'amplificateur. Comme les deux fonctionnent sur le même principe, concentrons-nous un instant sur $\hat{F}_i$. Cette fonction est exprimée :

$$\hat{F}_i(|x(k-\tau_i)|) = \sum_{p=0}^{P} \alpha_{pi}\,|x(k-\tau_i)|^p \tag{2.3}$$

où τ_i est le délai de la i^{ème} fonction polynomiale

α_{pi} est le coefficient du terme d'ordre p du i^{ème} polynôme

Si nous remplaçons (2.3) dans (2.1), nous obtenons donc :

$$f_i\big(x(k-\tau_i)\big) = \sum_{p=0}^{P} \alpha_{pi}\, x(k-\tau_i)|x(k-\tau_i)|^p \tag{2.4}$$

L'équation (2.4), dont on peut calculer l'équivalent pour g_j, est une série de puissances. Or, il est bien connu que les séries de puissances peuvent être utilisées pour représenter les effets non linéaires d'un amplificateur de puissance RF, ainsi que pour représenter la fonction de prédistorsion qui permet de linéariser cet amplificateur. La méthode de prédistorsion NARMA ne fait donc pas exception sur ce point.

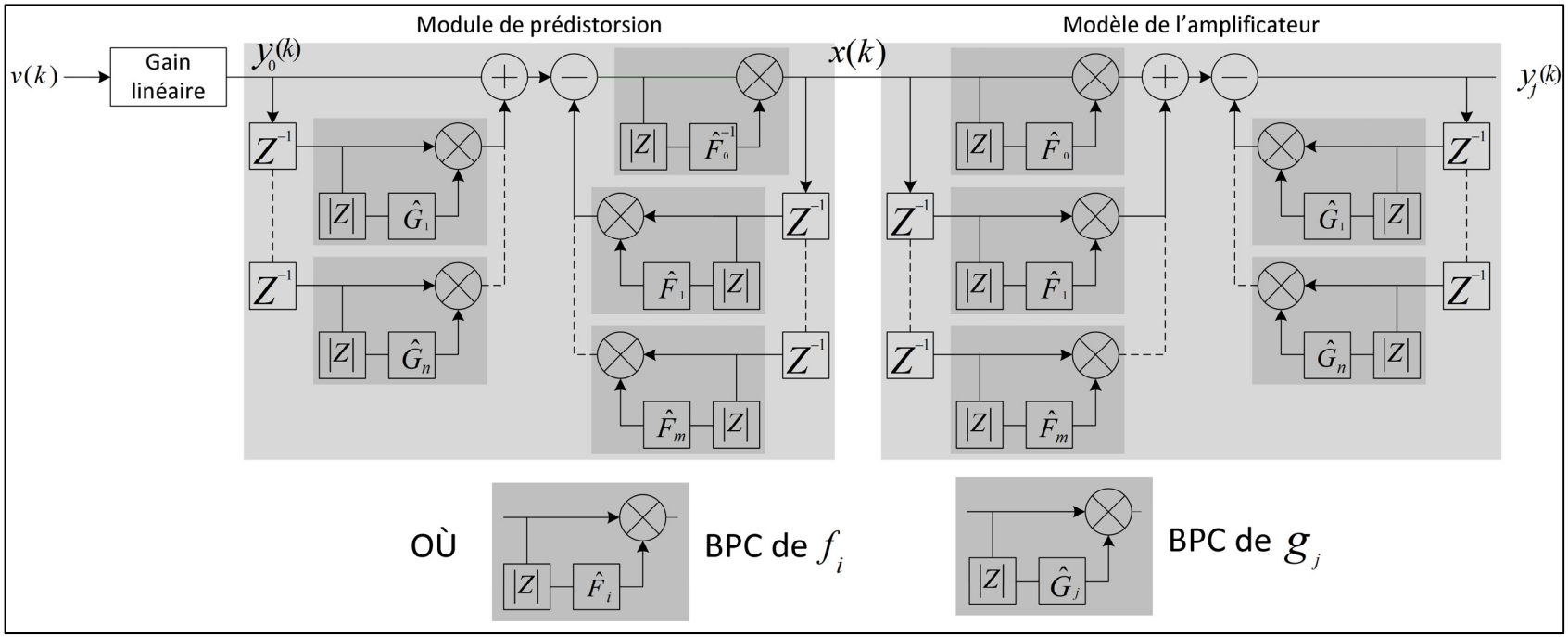


Figure 2.2 Diagramme détaillé de la combinaison d'une modélisation de type NARMA d'un amplificateur avec le module de prédistorsion servant à le linéariser

Analysons la relation entre l'entrée et la sortie :

$$x(k) = f_0^{-1}\left[y_0(k) + \sum_{j=1}^{n} g_j\left(y_0\left(k-\tau_j\right)\right) - \sum_{i=1}^{m} f_i\left(x(k-\tau_i)\right)\right] \tag{2.5}$$

et

$$y_f(k) = f_0\left(x(k)\right) + \sum_{i=1}^{m} f_i\left(x(k-\tau_i)\right) - \sum_{j=1}^{n} g_j\left(y_f\left(k-\tau_j\right)\right) \tag{2.6}$$

sont déduites directement de la Figure 2.2. On peut ensuite remplacer (2.5) dans (2.6), pour obtenir ceci :

$$\begin{aligned} y_f(k) = f_0\left(f_0^{-1}\left[y_0(k) + \sum_{j=1}^{n} g_j\left(y_0\left(k-\tau_j\right)\right) - \sum_{i=1}^{m} f_i\left(x(k-\tau_i)\right)\right]\right) \\ + \sum_{i=1}^{m} f_i\left(x(k-\tau_i)\right) - \sum_{j=1}^{n} g_j\left(y_f\left(k-\tau_j\right)\right) \end{aligned} \tag{2.7}$$

En utilisant la définition d'une fonction réciproque, soit $f_0(f_0^{-1}(h(z))) = h(z)$, et en simplifiant, nous trouvons l'équation (2.8). Celle-ci relie l'entrée et la sortie par une équation de forme implicite :

$$y_f(k) = y_0(k) + \sum_{j=1}^{n} g_j\left(y_0\left(k-\tau_j\right)\right) - \sum_{j=1}^{n} g_j\left(y_f\left(k-\tau_j\right)\right) \tag{2.8}$$

Dans (Gilabert *et al.*, 2008), les auteurs analysent plus en profondeur l'efficacité de la méthode, et évaluent les différents compromis pouvant être réalisés. Parmi les options paramétrables, deux des plus importantes sont le nombre de blocs du filtre FIR ou le nombre du filtre IIR en raison de leur impact sur l'efficacité de la linéarisation. On retrouve aussi l'évaluation des délais à utiliser entre les blocs. À la Figure 2.2, les différents BPC sont espacés d'un délai d'un cycle. Cette généralisation a été faite pour fins de simplification. Néanmoins, plusieurs blocs intermédiaires peuvent être nuls de façon à avoir par exemple, un

BPC du filtre FIR au délai 0, puis un autre avec un retard de 10 cycles. Autrement dit, on doit choisir le nombre de BPC à utiliser dans les filtres FIR et IIR, mais aussi on doit déterminer les décalages entre ces derniers, qui correspondent à des intervalles de temps. Ces choix doivent être faits dans l'objectif de réaliser la meilleure prédistorsion possible au moindre coût.

Différentes techniques ont été proposées afin d'obtenir le modèle NARMA de l'amplificateur. Dans (Gilabert *et al.*, 2008), c'est plutôt l'utilisation de l'algorithme LS (Least Squares) qui est favorisée car il permet de faire un meilleur usage de blocs d'échantillons de 2048 données. Les auteurs notent toutefois au passage la possibilité d'utiliser les algorithmes RLS (Recursive Least Squares) ou LMS (Least Mean Squares). Comme nous le verrons plus tard, c'est toutefois ce dernier algorithme qui a été sélectionné dans le cadre de ce travail, en raison de sa simplicité d'implémentation et à cause des détails précis fournis par (Ning *et al.*, 2010), au sujet de son utilisation avec un modèle NARMA, dont les fonctions non linéaires sont des polynômes complexes.

2.2.3 Prédistorsion NARMA (v2)

Dans (Gilabert, Montoro et Bertran, 2011), les auteurs suggèrent une autre implémentation basée sur une structure de prédistorsion de type NARMA. Afin d'alléger la lecture, cette implémentation sera référée en tant que NARMA v2. Il s'agit d'une implémentation temps-réel basée sur la méthode proposée dans (Gilabert, Montoro et Cesari, 2006). Contrairement à NARMA v1, on ne procède pas par inversion d'un modèle NARMA de l'amplificateur. En fait, ce modèle n'est pas utilisé du tout.

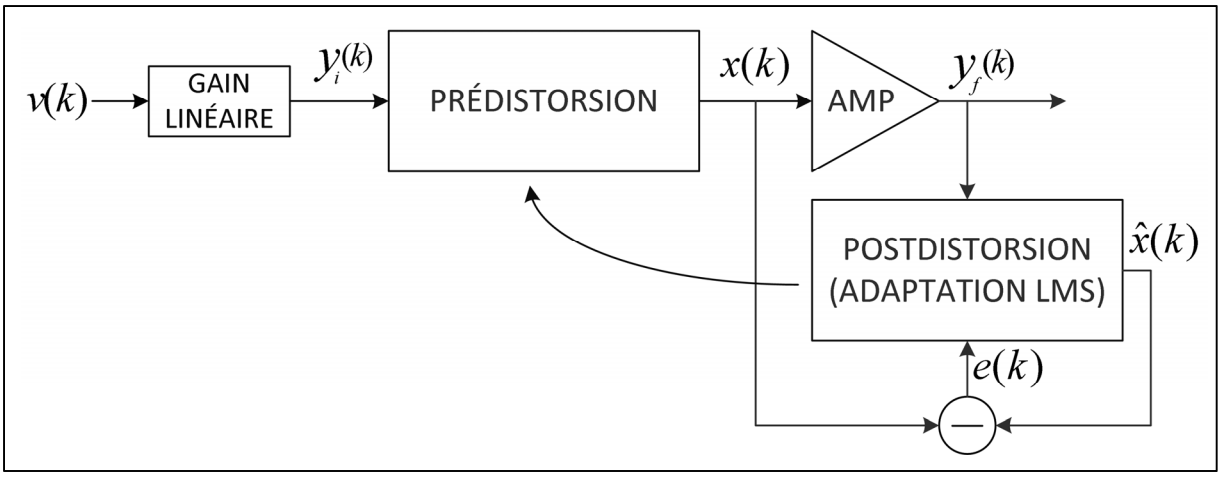


Figure 2.3 Principe de la prédistorsion avec structure NARMA (v2)

Dans NARMA v2 (Voir Figure 2.3), les auteurs procèdent directement à la caractérisation du module de prédistorsion. Afin d'y parvenir, ils définissent une structure de type NARMA comme module de postdistorsion. Cette structure prend les entrées/sorties de l'amplificateur inversées, et utilise un algorithme LMS pour calculer un modèle qui réalise la fonction inverse. Le module de postdistorsion ainsi calculé est donc ultimement utilisé en prédistorsion. Le plus grand avantage de cette approche est qu'il n'est pas nécessaire de calculer un modèle de l'amplificateur ainsi que de procéder à son inversion. Il s'agit d'un apprentissage de type indirect, car on ne modifie pas directement le module de prédistorsion en boucle fermée. Dans la citation ci-dessous, tirée d'un autre article, les auteurs expliquent ce type d'apprentissage :

> Once the learning process has finished, the postdistorter coefficients are copied to an identical model that is used to predistort the input. With this configuration there is no need to calculate the PA model, only its input and output data is needed to estimate the postdistortion model. (Gilabert, Montoro et Bertran, 2005)

2.3 Applicabilité des méthodes NARMA dans un milieu de développement

Les techniques de prédistorsion NARMA présentées aux sections précédentes ont l'avantage d'être facilement applicables dans un milieu de développement. En effet, dans un tel contexte, il est possible que l'amplificateur du système développé soit prêt à être testé avant que l'ensemble du système ne le soit. Les méthodes de prédistorsion NARMA présentées ne nécessitent que de valeurs d'entrée/sortie de l'amplificateur en question afin de pouvoir être utilisées. Ces données peuvent être obtenues relativement tôt dans le cycle de développement, notamment au moyen d'outils de simulation comme ADS. Aussi, dès que l'amplificateur est physiquement disponible, on peut, au moyen d'un générateur de signaux vectoriels et d'un analyseur de signaux vectoriels, obtenir ces données de façon expérimentale.

Pendant le développement d'un système de transmission, il peut arriver que ce soit l'architecture de calcul NARMA qui ne soit pas encore réalisée physiquement ou fonctionnelle. Dans le cas de NARMA v1, on parle de l'architecture permettant de calculer le modèle de l'amplificateur et de procéder à son inversion, tandis que dans le cas de NARMA v2, on parle de celle qui calcule directement le module de prédistorsion. Parallèlement à ce développement, si on souhaite valider l'efficacité de ces méthodes de prédistorsion sur l'amplificateur, il est possible de réaliser l'architecture de calcul sur un PC. Celle-ci peut être exécutée, et le module de prédistorsion calculé peut ensuite être utilisé pour modifier les données générées par le générateur de signaux vectoriels. On peut donc rapidement procéder à la validation de l'efficacité de la technique de prédistorsion.

Si on compare la puissance de calcul nécessaire, entre NARMA v1 et NARMA v2, pour calculer un module de prédistorsion, la seconde version est probablement moins coûteuse en ce sens que la première. Ceci s'explique par la nécessité pour NARMA v1 d'inverser la fonction f_0. Néanmoins, NARMA v1 présente un avantage intéressant : on se trouve à calculer un modèle de l'amplificateur. Ce modèle peut ensuite être réutilisé, par exemple dans le cas où on souhaite facilement réaliser une simulation MATLAB de l'amplificateur.

Dans le chapitre suivant, nous verrons aussi que NARMA v1 permet de calculer un module de prédistorsion plus facilement, dans le cas où l'amplificateur a une réponse non monotone.

2.4 Accélération de l'adaptation NARMA

Dans le contexte de l'utilisation de NARMA (v1 ou v2) sur un appareil mobile, il peut être avantageux d'initialiser le système au moyen de valeurs précalculées afin d'accélérer le processus d'adaptation. Cette idée est illustrée à la Figure 2.4 pour le cas de NARMA v1 : le module qui réalise la modélisation de l'amplificateur au moyen de LMS peut démarrer son adaptation à partir d'un modèle présent en mémoire.

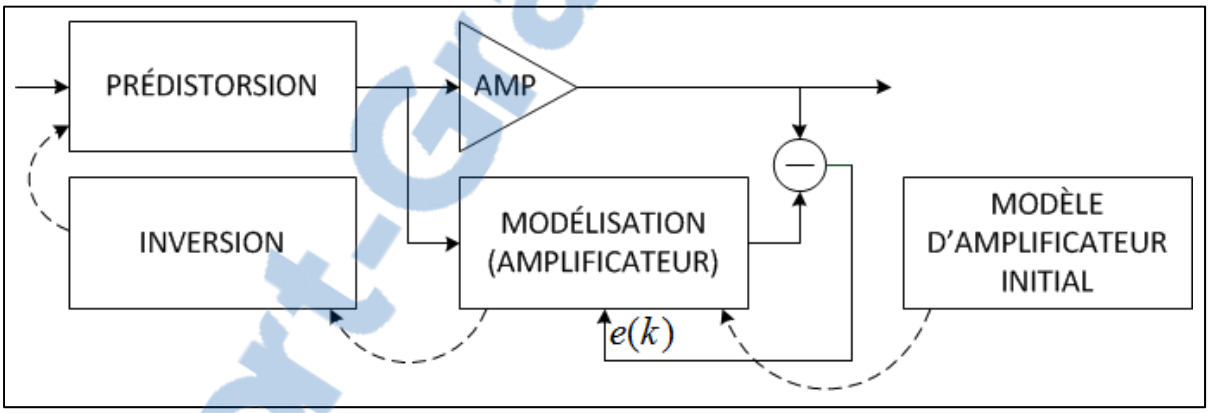


Figure 2.4 Initialisation de NARMA (v1)

On peut procéder de façon équivalente pour NARMA v2, tel qu'illustré à la Figure 2.5. Contrairement à l'exemple précédent, il faut cependant noter que l'initialisation est faite à partir d'un module de prédistorsion puisqu'il n'y a pas d'inversion dans ce cas.

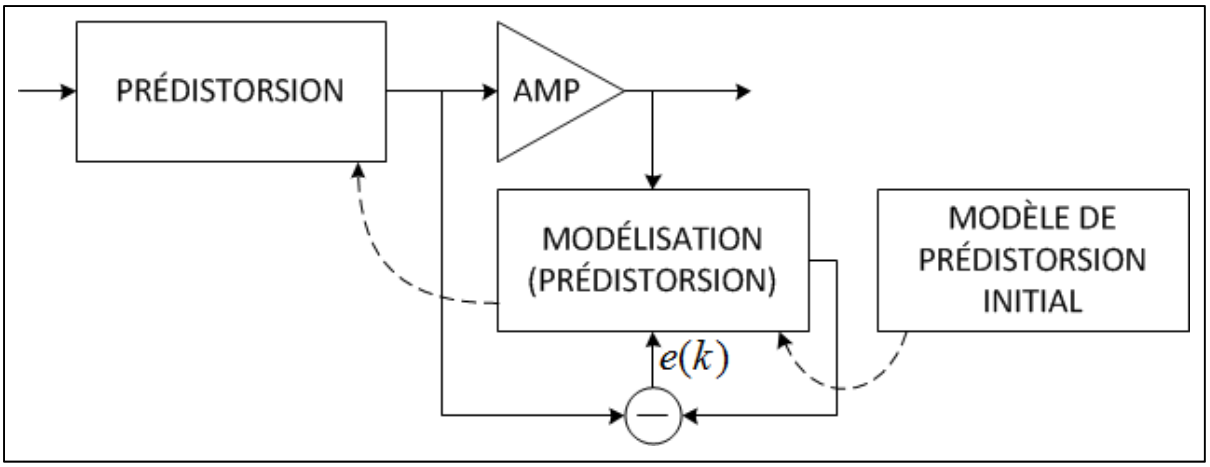


Figure 2.5 Initialisation de NARMA (v2)

Dans un cas comme dans l'autre, les valeurs initiales peuvent avoir été calculées en laboratoire, par exemple, durant la phase de développement.

2.5 Conclusion

Dans ce chapitre, nous avons d'abord vu la notion de linéarité dans un amplificateur de puissance et son importance du point de vue de l'efficacité énergétique. Nous avons enchaîné avec l'effet mémoire pour ensuite faire un survol de la reconfiguration hardware dans les amplificateurs. Par la suite, nous avons vu différentes techniques de linéarisation par prédistorsion logicielle. Finalement, deux techniques de prédistorsion NARMA ont été étudiées dans les contextes d'un environnement de développement et de l'application en communications sans fil.

CHAPITRE 3

PRÉDISTORSION D'UN AMPLIFICATEUR AVEC CARACTÉRISTIQUE NON MONOTONE

3.1 Introduction

Cette section traite de la linéarisation d'un amplificateur ayant une caractéristique non monotone au moyen d'un module de prédistorsion ayant une configuration de type NARMA. Ce sujet a été l'objet de l'article (Thibodeau, Kouki et Constantin, 2013), publié dans le cadre de la conférence SiRF d'IEEE qui s'est tenue à Austin en janvier 2013.

Dans un premier temps, nous verrons l'architecture matérielle de l'amplificateur reconfigurable, puis nous étudierons sa caractéristique entrée/sortie. Ensuite, nous verrons en quoi les limitations du filtre LMS inversé ainsi que d'autres techniques posent problème dans le cas considéré. Finalement, la technique suggérée sera expliquée, appliquée à la méthode NARMA puis validée.

3.2 Architecture matérielle de l'amplificateur reconfigurable

L'amplificateur reconfigurable considéré est celui dont l'architecture matérielle est illustrée à la Figure 3.1. L'amplificateur est conçu pour être reconfiguré électroniquement, selon qu'il opère en régime petit-signal ou en régime grand-signal, avec une transition graduelle entre les deux régimes, en fonction de l'amplitude instantanée de l'enveloppe du signal RF amplifié.

On retrouve deux cellules de transistors RF, soit C1 et C8. La cellule C1 est constituée d'un seul transistor. La cellule C8, pour sa part, consiste en un assemblage de huit transistors en parallèle. Dans tous les cas, le modèle de transistor est le HBFP-0450 (Voir ANNEXE I, p.87). Ce transistor a été choisi en raison de la disponibilité des modèles linéaires et non linéaires dans le logiciel ADS.

À l'entrée, on retrouve le circuit d'adaptation d'impédance d'entrée MI9. Le circuit MO9 joue un rôle équivalent en sortie. À haute puissance, les cellules C1 et C8 sont toutes deux en opération. Les circuits MI9 et MO9 sont conçus afin d'assurer que, dans ces conditions, la puissance en sortie soit maximale et que le gain de la puissance réfléchie (en anglais : « Return Loss ») reste sous la barre des -25 dB. Il s'agit alors d'une condition d'adaptation d'impédance pour une puissance de sortie maximisée en régime grand-signal. Les circuits MI9 et MO9 sont fixes, et ne varient pas en fonction de la puissance.

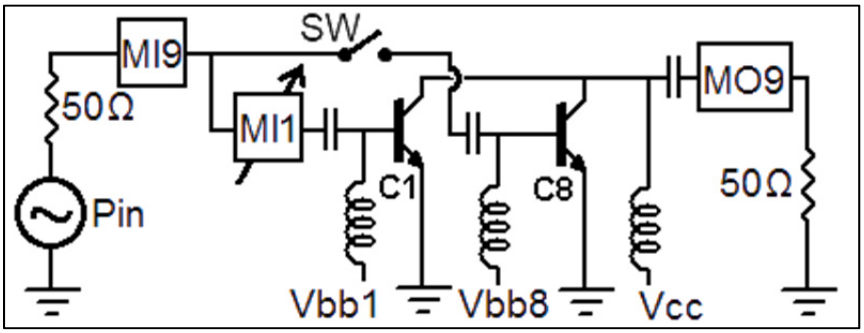


Figure 3.1 Architecture de l'amplificateur
de puissance reconfigurable

À faible puissance, seule la cellule C1 est active, afin d'avoir une consommation de courant minimale, pour une meilleure efficacité énergétique. Son courant de polarisation est alors ajusté à 80 mA à travers la tension de contrôle Vbb1, afin de maintenir le gain RF désiré à basse puissance. Le circuit d'adaptation d'impédance MI1 est alors ajusté pour une condition d'adaptation d'impédance favorisant un gain RF maximisé en régime petit-signal.

Lorsque l'amplitude de l'enveloppe augmente à partir d'environ 0,7 V à 1,8 V, l'amplificateur entame une transition progressive à partir de la condition d'adaptation d'impédance pour un gain RF maximisé en régime petit-signal vers la condition d'adaptation d'impédance pour une puissance de sortie maximisée en régime grand-signal. Pour cela, le courant de polarisation total de C8 est progressivement augmenté à l'aide de la tension de contrôle Vbb8 jusqu'à ce qu'il atteigne une valeur d'environ 160 mA, soit 20 mA pour chaque transistor de la cellule. En même temps, le courant de polarisation du transistor C1 est

progressivement diminué jusqu'à 20 mA à l'aide de la tension de contrôle Vbb1 et le circuit d'adaptation MI1 est ajusté pour favoriser le régime grand-signal. L'interrupteur SW est aussi graduellement fermé en ajustant un atténuateur électronique à faible perte.

À haute puissance, l'interrupteur SW est fermé tandis que les transistors C1 et C8 sont activés. Les circuits MI9 et MO9 atteignent leur maximum d'efficacité puisqu'ils sont conçus pour cette condition d'opération, tel qu'expliqué précédemment.

3.3 Caractéristique entrée/sortie non monotone

Le circuit reconfigurable illustré à la Figure 3.1 a été simulé avec une excitation de type CW dont l'amplitude varie sur une plage donnée, et de fréquence 1,88 GHz. La Figure 3.2. montre la réponse simulée AM-AM entrée/sortie du circuit. On y voit trois courbes. D'abord, on retrouve la courbe de faible puissance, qui décrit la caractéristique entrée/sortie dans le cas où le circuit est dans sa configuration pour le régime petit-signal, où le gain RF est maximisé. On peut noter sur la figure que si la configuration restait la même et que la tension d'entrée continuait d'augmenter, le gain en tension chuterait. Ceci s'explique par le fait que la cellule C1 entrerait alors en région de compression et ne parviendrait plus à délivrer de puissance additionnelle qu'au coût de plus en plus important d'une puissance d'entrée grandissante. C'est donc à ce point que s'entame la transition vers l'état de haute puissance.

À partir d'environ 0.7 V de tension d'entrée, le circuit entame graduellement sa reconfiguration pour le régime grand-signal. Cette transition entraîne une diminution de la tension de sortie entre les deux états, à cause de conditions hardware non-désirées d'adaptation d'impédance et de polarisation qui transitent par des valeurs non-optimales. La courbe de transition à la Figure 3.2 représente le cas où, pendant la transition, la réponse en tension diminue le plus. Évidemment, pour obtenir cette courbe de transition comme illustrée dans la figure, on doit fixer l'état de MI1 et SW. La réponse réelle de l'amplificateur est en fait le déplacement sur les différentes courbes intermédiaires (non illustrées), le long de la courbe de réponse résultante illustrée dans la figure, pendant que la reconfiguration se

produit. Aux tensions d'entrée plus élevées, la courbe de réponse résultante tend à rejoindre la courbe de haute puissance, lorsque la reconfiguration pour le régime grand-signal est atteinte. On remarque que la réponse AM-AM le long de la courbe résultante devient non monotone pendant la reconfiguration. Cette caractéristique, comme nous le verrons bientôt, pose un problème particulier si on souhaite linéariser l'amplificateur au moyen d'une prédistorsion de type NARMA.

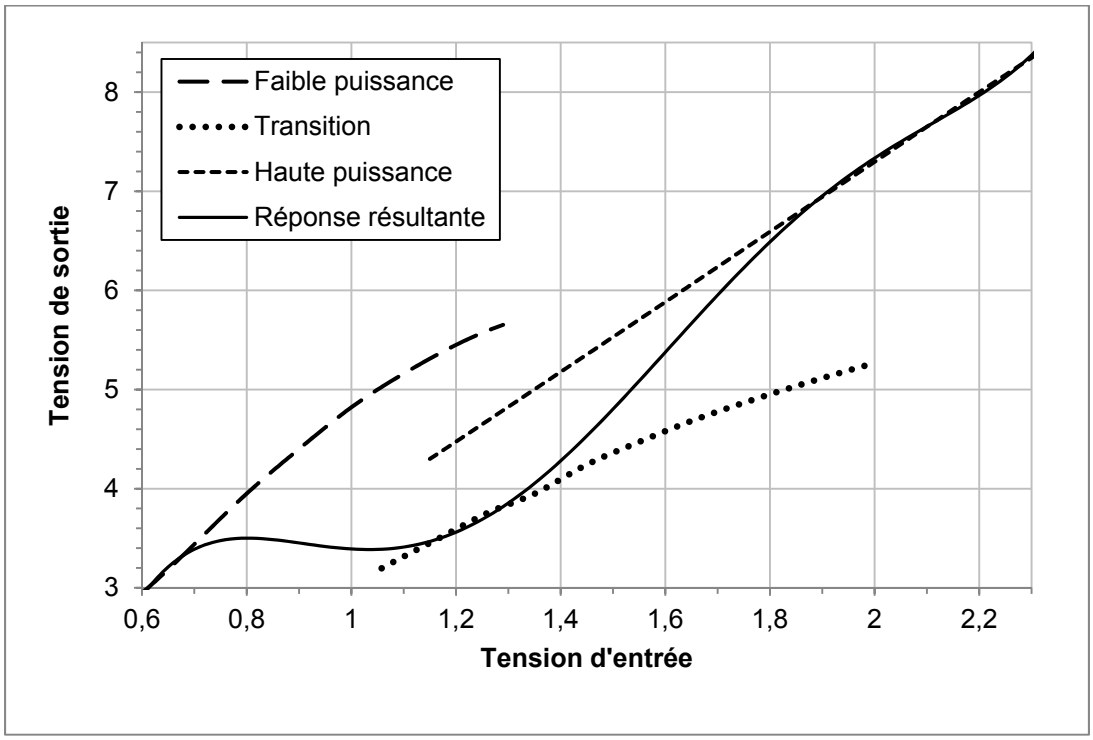


Figure 3.2 Réponse simulée des tensions d'entrée/sortie de l'enveloppe pour l'amplificateur de puissance reconfigurable

3.4 Inefficacité d'un filtre LMS inversé

Tel que vu dans la section précédente, un amplificateur reconfigurable peut présenter une caractéristique AM-AM non monotone. L'implémentation réalisée dans (Gilabert *et al.*, 2008) consiste, à partir d'un grand nombre de couples de tensions entrée/sortie, à identifier les fonctions polynomiales des différents BPC du modèle NARMA. La caractéristique non monotone de la courbe AM-AM fera éventuellement converger le polynôme de la cellule f_0

vers une fonction non monotone. Il n'est pas problématique d'avoir une fonction non monotone à ce stade, car les algorithmes permettant d'obtenir le modèle NARMA de l'amplificateur sont en mesure de converger sur cette réponse.

La difficulté survient lorsque vient le temps de calculer le module de prédistorsion correspondant. Pour procéder à l'inversion du modèle, tous les BPC, à l'exception de celui de f_0, sont réutilisés directement. Pour compléter l'inversion, il faut néanmoins trouver la réciproque mathématique de la fonction f_0. La méthode suggérée dans (Montoro *et al.*, 2007), consiste à réutiliser les données de caractérisation en sens inverse. Au moyen d'un algorithme comme LMS, on peut donc parvenir à trouver le contenu de la fonction f_0^{-1}. Toutefois, cela suppose qu'on peut décrire la réciproque comme une fonction. Or, pour que la fonction réciproque d'une fonction existe, il nécessaire que cette fonction soit strictement non monotone. Si cette condition n'est pas respectée, comme on peut le voir à la Figure 3.3, la réciproque n'est pas une fonction, puisqu'on peut retrouver plus d'une ordonnée sur une même abscisse.

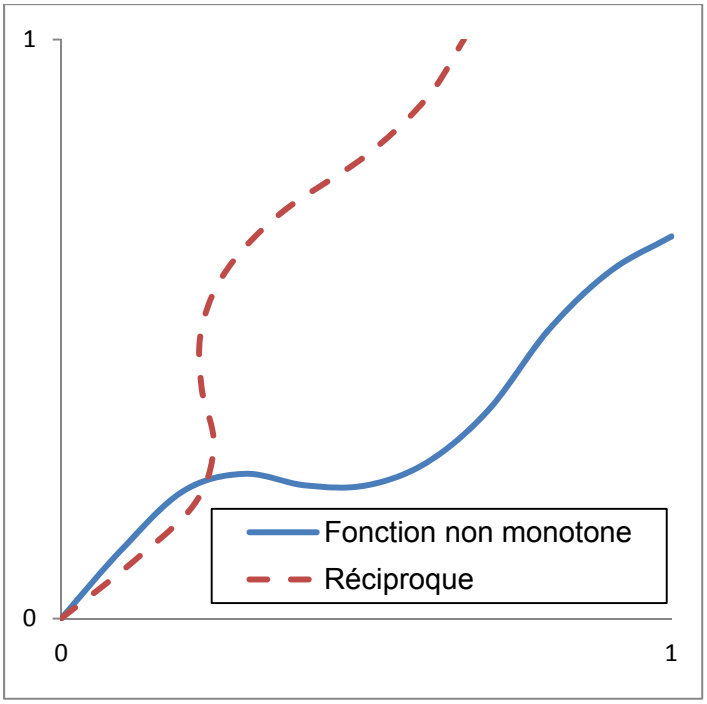


Figure 3.3 Réciproque d'une fonction non monotone

De toute évidence, une fonction polynomiale ne peut en aucun cas permettre d'exprimer une courbe qui n'est pas définie par une fonction. La technique décrite, basée sur LMS, est donc mal adaptée pour traiter le cas où la fonction f_0 n'est pas monotone stricte.

3.5 Inefficacité de l'utilisation d'une équation implicite

Dans (Ning *et al.*, 2010), une technique différente est proposée afin de trouver la réciproque de f_0. Au terme de leur analyse mathématique, les auteurs proposent d'utiliser l'équation suivante :

$$G_{LUT_{f_0^{-1}}}(|z(k)|) = \frac{1}{\sum_{p=0}^{P} \alpha_{p0} \cdot \left(|z(k)| \cdot \left|G_{LUT_{f_0^{-1}}}(|z(k)|)\right|\right)^p} \tag{3.1}$$

Ainsi, en faisant varier $|z(k)|$, on obtient une relation dans laquelle il ne reste plus qu'à isoler $G_{LUT_{f_0^{-1}}}(|z(k)|)$. C'est néanmoins là que le problème se pose. L'équation (3.1) est une équation implicite. Résoudre ce genre d'équation est loin d'être trivial. Il faut généralement compter sur des algorithmes de recherche qui peuvent s'avérer coûteux en termes de puissance de calcul. Ce qui rend ce calcul encore plus complexe, c'est que dans le cas de la réciproque d'une fonction non monotone, il y a plusieurs valeurs qui permettent de résoudre l'équation. S'assurer de trouver tous les couples de solution devient alors crucial afin de décider de la valeur optimale à utiliser. Il n'y a toutefois pas d'indication claire qui permettrait de savoir si tous les couples ont été trouvés ou non. C'est donc pour ces raisons que cette approche a été rejetée dans ce présent travail.

3.6 Introduction de l'équation explicite proposée dans ce travail

Étant donné la déficience des méthodes mentionnées aux sections précédentes pour obtenir de façon déterministe et rapide la réciproque de la fonction non strictement monotone f_0, une nouvelle analyse mathématique du problème a été effectuée et a donné lieu à une nouvelle méthode de calcul. Celle-ci a d'ailleurs fait l'objet d'une publication (Voir ANNEXE II,

p.91). Concentrons-nous particulièrement sur la fonction f_0 et sa réciproque, illustrés à la Figure 3.4, et analysons l'interaction entre ces deux blocs.

3.6.1 Analyse

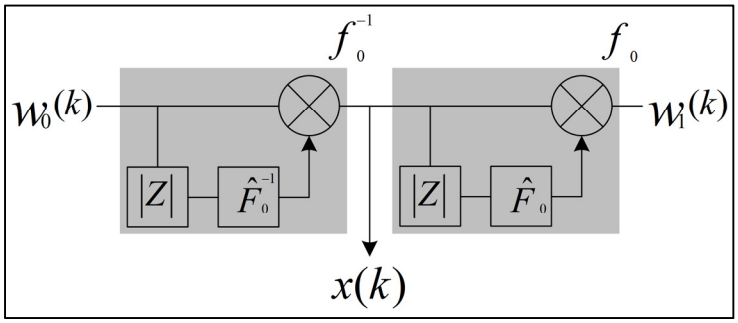


Figure 3.4 Diagramme des blocs d'une fonction précédée de sa réciproque

En assumant que le premier bloc est la réciproque du second, on peut affirmer que $w_0(k) = w_1(k)$. Nous pouvons établir les deux équations suivantes :

$$w_1(k) = w_0(k) = f_0\big(x(k)\big) \tag{3.2}$$

$$x(k) = f_0^{-1}\big(w_0(k)\big) \tag{3.3}$$

On peut ensuite procéder à l'expansion de ces équations en tenant compte du contenu des blocs des fonctions f_0 et f_0^{-1}.

$$w_1 = x(k)\hat{F}_0(|x(k)|) \tag{3.4}$$

$$x(k) = w_0(k)\hat{F}_0^{-1}(|w_0(k)|) \tag{3.5}$$

En considérant que $w_0(k) = w_1(k)$ et en substituant l'équation (3.4) dans l'équation (3.5), nous obtenons:

$$x(k) = x(k)\hat{F}_0(|x(k)|)\hat{F}_0^{-1}\big(\big|x(k)\hat{F}_0(|x(k)|)\big|\big) \tag{3.6}$$

On peut ensuite diviser par $x(k)\hat{F}_0(|x(k)|)$ de chaque côté pour obtenir :

$$\frac{1}{\hat{F}_0(|x(k)|)} = \hat{F}_0^{-1}\left(|x(k)|\left|\hat{F}_0(|x(k)|)\right|\right) \tag{3.7}$$

Afin de représenter les valeurs absolues dans un format de variable scalaire plus près de l'implémentation algorithmique réalisée plus tard, faisons les substitutions de variables suivantes :

$$r = |x(k)| \tag{3.8}$$

$$h(r) = r\left|\hat{F}_0(r)\right| \tag{3.9}$$

Finalement, en remplaçant dans (3.7) nous obtenons :

$$\hat{F}_0^{-1}\left(r\left|\hat{F}_0(r)\right|\right) = \hat{F}_0^{-1}\big(h(r)\big) = \frac{1}{\hat{F}_0(r)} \tag{3.10}$$

Contrairement à l'approche de (Ning *et al.*, 2010), la relation obtenue en (3.10) est une équation explicite en termes de $\hat{F}_0^{-1}$, étant donné que $\hat{F}_0(r)$ est trouvée lors de l'étape de caractérisation au moyen de l'algorithme LMS.

Ainsi, pour toute valeur de r que l'on fixe, nous pouvons calculer le paramètre de $\hat{F}_0^{-1}$ et sa valeur résultante. Cette situation présente déjà le grand avantage de ne pas nécessiter de tester une multitude de couples de valeurs pour valider l'équation : une seule valeur doit être fixée pour obtenir un résultat. Cela est vrai, que $\hat{F}_0(r)$ soit monotone ou non.

3.6.2 Recherche de la fonction inverse pour minimiser l'intensité de l'excitation de l'amplificateur

Nous étudions maintenant les relations qui permettent d'obtenir la fonction inverse $\hat{F}_0^{-1}$ assurant l'intensité $r = |x(k)|$ (Figure 3.4) la plus faible possible pour le signal d'entrée, lorsque survient un comportement de f_0 non-monotone dans la réponse de l'amplificateur. L'intérêt de minimiser l'intensité de l'excitation découle du fait que généralement une plus faible amplitude implique une plus faible consommation de courant dans l'amplificateur, et

donc une meilleure efficacité énergétique. On considère donc une courbe AM-AM non monotone et $h(r)$ l'est aussi. Nous devrons évaluer l'équation (3.10) en faisant varier r de façon continue et croissante, ce qui entraîne une variation croissante à la sortie, jusqu'à un certain moment où $h(r)$ se met à diminuer (voir comme exemple la Figure 3.2, entre 0.7 V et 1.1 V de tension d'entrée). Il est donc plausible de supposer que certaines valeurs de $\hat{F}_0^{-1}$ seront définies plus d'une fois. Cela traduit le fait que l'équation (3.10) n'est pas une fonction, mais une définition paramétrique de la réciproque de la fonction multiplicative de f_0. Cette définition a toutefois une autre propriété intéressante que nous exploiterons afin de définir notre fonction finale.

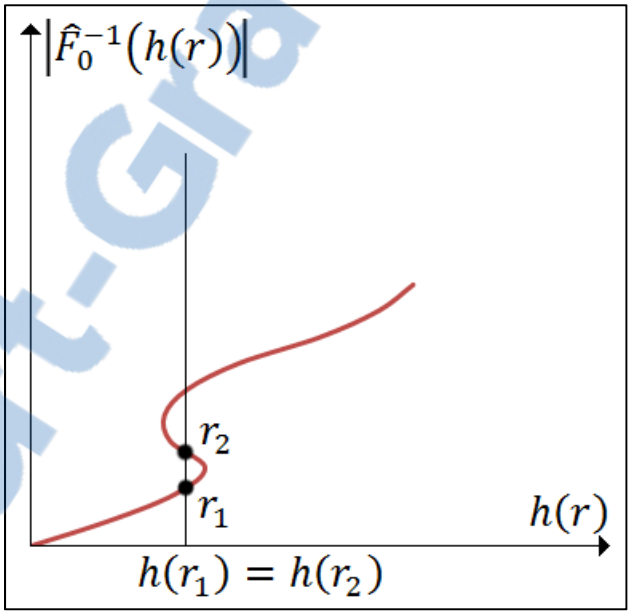


Figure 3.5 Évaluation de $\hat{F}_0^{-1}$ lorsque r est monotone croissante stricte

Considérons le cas de la Figure 3.5, où deux valeurs consécutives de r, soit r_1 et r_2 avec $r_1 < r_2$, sont utilisées pour évaluer le module de $\hat{F}_0^{-1}$ au moyen de l'équation (3.10).

$$r_1 < r_2 \tag{3.11}$$

Voyons particulièrement le cas où deux abscisses consécutives se superposent, soit lorsque $h(r_1) = h(r_2)$. Commençons par évaluer la fonction h en ces deux points en nous basant sur la définition (3.9):

$$h(r_1) = h(r_2) \Rightarrow r_1\left|\hat{F}_0(r_1)\right| = r_2\left|\hat{F}_0(r_2)\right| \tag{3.12}$$

En remplaçant l'équation (3.12) dans l'inégalité (3.11), nous obtenons:

$$\left|\hat{F}_0(r_1)\right| > \left|\hat{F}_0(r_2)\right| \tag{3.13}$$

En prenant le module de l'équation (3.10), nous obtenons :

$$\left|\hat{F}_0^{-1}\big(h(r)\big)\right| = \frac{1}{\left|\hat{F}_0(r)\right|} \tag{3.14}$$

Finalement, en remplaçant (3.14) dans l'inégalité (3.13) en prenant soin d'ajuster les indices de r correctement, nous obtenons :

$$\left|\hat{F}_0^{-1}\big(h(r_1)\big)\right| < \left|\hat{F}_0^{-1}\big(h(r_2)\big)\right| \tag{3.15}$$

Le résultat (3.15) est important car il indique que lorsque r croît à partir de sa valeur minimale, le premier passage au-dessus d'une abscisse (point correspondant r_1 à la Figure 3.5) donne toujours la valeur la plus faible de $\hat{F}_0^{-1}$, pour l'abscisse considérée. Ainsi, en faisant varier r de façon croissante à partir de la plus faible amplitude envisageable vers la plus grande, on calcule directement la réciproque de façon paramétrique. Lorsque $h(r)$ diminue, on peut cesser la capture des données jusqu'à ce que $h(r)$ augmente à nouveau au-delà du point où il avait commencé à diminuer. En utilisant cette approche systématique, on choisit toujours les valeurs minimales de $\left|\hat{F}_0^{-1}\big(h(r)\big)\right|$ qui permettent de linéariser l'amplificateur. Finalement, les résultats sont stockés dans la table de correspondance pour toute la plage de valeurs de $h(r)$ et peuvent être utilisés pour la linéarisation.

3.7 Application de la prédistorsion NARMA

Le cheminement mathématique développé à la section 3.6 s'inscrit comme un élément de la chaîne d'opérations à effectuer afin de pouvoir réaliser une prédistorsion de type NARMA qui permet de linéariser la réponse de l'amplificateur reconfigurable. Pour cette fin, un modèle de simulation dans MATLAB a été créé.

3.7.1 Génération de données de caractérisation

Dans un premier temps, il a été nécessaire de créer un modèle de l'amplificateur qui soit utilisable dans MATLAB. L'aspect qui nous intéresse ici, c'est la prédistorsion du module plus que de la phase, étant donné que c'est lui qui crée la non monotonie à l'origine des difficultés mentionnées dans les sections précédentes. Pour cette raison, les effets mémoire ont été négligés puisque de toute façon, les auteurs de (Gilabert *et al.*, 2008) ont déjà démontré l'efficacité de la méthode NARMA par rapport à ceux-ci. En ce qui concerne la phase, une courbe typique a donc été utilisée dans le modèle, basée sur des simulations de circuits amplificateurs.

D'abord, en utilisant plusieurs points de la courbe AM-AM de l'amplificateur, une courbe polynomiale équivalente a été calculée. Une fois cette courbe établie, l'étape suivante est de générer des données d'entrée/sortie en simulation. C'est ce que permet le modèle Simulink de la Figure 3.6. Deux sources de données I/Q peuvent être utilisées. La première est un générateur de type 16-QAM dont la constellation peut être ajustée selon l'amplitude désirée. Il est toutefois essentiel de faire varier cette amplitude afin d'obtenir des données de caractérisation qui couvrent l'ensemble de la plage de valeurs de l'amplificateur. L'autre façon de générer des données consiste simplement à générer des valeurs d'amplitude et de phase aléatoires. Ces valeurs sont donc combinées et transformées en nombre complexe. Dans un cas comme dans l'autre, les données sont acheminées au modèle de l'amplificateur, qui génère une sortie qui contient de la distorsion. Les données d'entrée/sortie sont ensuite stockées pour une utilisation ultérieure. Cette procédure correspond donc exactement à celle

qui serait utilisée dans un contexte de laboratoire pour obtenir les données d'entrée/sortie. Évidemment, la différence ici, c'est que l'amplificateur est simulé.

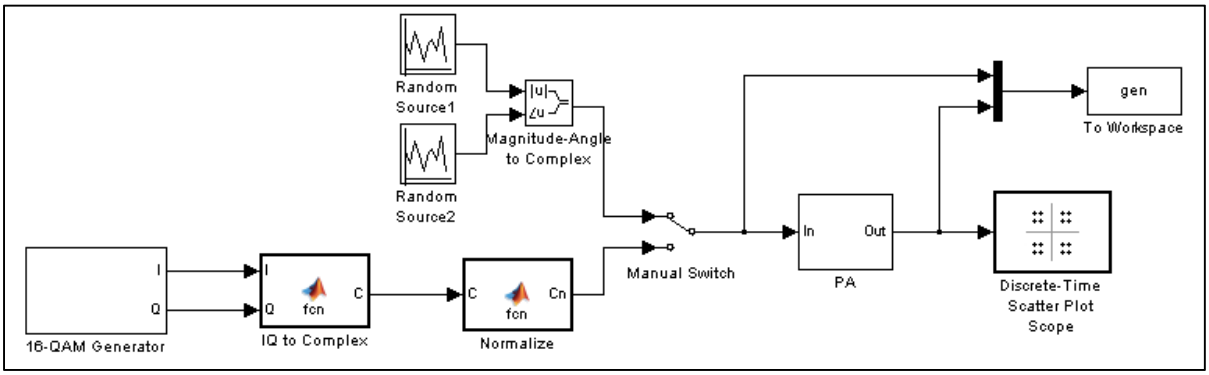


Figure 3.6 Modèle Simulink utilisé pour générer des données de caractérisation

3.7.2 Calcul du modèle avec LMS

Lors de l'étape précédente, plusieurs milliers de données de caractérisation ont été générées. Il s'agit de couples entrée/sortie de l'amplificateur. L'étape suivante consiste à analyser ces données au moyen de l'algorithme LMS pour calculer le modèle polynomial NARMA de l'amplificateur ($\hat{F}_0$ à la Figure 3.4). Il est donc utile ici de rappeler les formulations pertinentes pour l'implémentation de l'algorithme LMS présenté en ANNEXE III, p. 95, en revoyant les équations suivantes, qui sont tirées de (Ning *et al.*, 2010). On commence d'abord avec la définition des termes:

$$x_A^{pi}(k) = x_A(k-\tau_i)|x_A(k-\tau_i)|^p \tag{3.16}$$

$$y_A^{pj}(k) = y_A(k-\tau_j)\left|y_A(k-\tau_j)\right|^p \tag{3.17}$$

$$\begin{aligned}\boldsymbol{u}(k) = [&x_A^{00}(k), \dots, x_A^{P0}(k), \dots, x_A^{0N}(k), \dots x_A^{PN}(k),\\ &y_A^{01}(k), \dots, y_A^{P1}(k), \dots, y_A^{0D}(k), \dots, y_A^{PD}(k)]\end{aligned} \tag{3.18}$$

$$\widehat{\boldsymbol{\theta}} = [\alpha_{00}, \dots, \alpha_{P0}, \dots, \alpha_{0N}, \dots, \alpha_{PN}, -\beta_{01}, \dots, -\beta_{P1}, \dots, -\beta_{0D}, \dots, -\beta_{PD}]^T \tag{3.19}$$

où $x_A(k)$ est la k-ième entrée du filtre

$y_A(k)$ est la k-ième sortie du filtre

τ est un retard, en nombre de cycles, de l'entrée d'un filtre

p est l'ordre du coefficient

i est l'index de retard d'un polynôme FIR

j est l'index de retard d'un polynôme IIR

P est l'ordre des polynômes

N est l'ordre du filtre IIR

D est l'ordre du filtre FIR

Le vecteur $\widehat{\boldsymbol{\theta}}$ contient les coefficients complexes qui définissent les fonctions non linéaires f_i et g_i de la structure NARMA du modèle de l'amplificateur. Quant au vecteur $\boldsymbol{u}(k)$, il s'agit d'une concaténation des valeurs $x_A^{pi}(k)$ ou $y_A^{pj}(k)$ qui correspondent à chacun des coefficients des polynômes consécutifs des filtres FIR puis IIR. Une fois ces variables calculées, on itère pour déterminer les prochaines valeurs des coefficients en fonction de l'erreur obtenue.

$$\hat{y}_A(k) = \widehat{\boldsymbol{\theta}}^H(k)\boldsymbol{u}(k) \tag{3.20}$$

$$e(k) = y_A(k) - \hat{y}_A(k) \tag{3.21}$$

$$\widehat{\boldsymbol{\theta}}(k+1) = \widehat{\boldsymbol{\theta}}(k) + \mu\boldsymbol{u}(k)e^*(k) \tag{3.22}$$

où $e(k)$ est l'erreur

$\widehat{\boldsymbol{\theta}}(k+1)$ est la prochaine estimation des coefficients

μ est le facteur de convergence

Dans l'équation (3.20), on note la lettre H en exposant : c'est la transposée hermitienne, qui consiste à transposer la matrice et à trouver le complexe conjugué de chaque terme. On comprend qu'il s'agit ici d'une représentation vectorielle d'une structure d'adaptation LMS, qui pourrait aussi être implémentée de façon plus visuelle dans Simulink. L'implémentation complète de l'algorithme LMS, codée pour MATLAB, est disponible en annexe (Voir ANNEXE III, p.95).

3.7.3 Calcul de la réciproque avec l'équation explicite proposée

Une fois l'ensemble des blocs du modèle NARMA de l'amplificateur caractérisé avec l'algorithme LMS, l'étape suivante consiste à utiliser l'équation explicite pour obtenir la réciproque de la fonction f_0. En fait, nous cherchons directement les données à insérer dans la table de correspondance $\hat{F}_0^{-1}$. C'est l'Algorithme 3.1 qui est exécuté à cette fin. Son implémentation pour MATLAB est disponible en annexe (Voir ANNEXE IV, p.97).

Algorithme 3.1 Calcul de la table de correspondance de la réciproque

```
1     Initialize Bounds and Step Size
2     while h is smaller than the upper bound
3           curr_r = r + dr
4           if h(curr_r) increased too much
5                 Reduce dr
6                 continue
7           end if
8           if h(curr_r) is too close to its value when we last stored data
9                 r = curr_r
10                continue
11          end if
12          Increase dr
13          if h(curr_r) is greater than its value when we last stored data
14                Calculate the LUT value and store it
15          end if
16    end while
```

Au terme de cette étape, nous disposons de toutes les données nécessaires à l'application d'une prédistorsion de type NARMA puisque les autres tables de correspondance utilisent directement celles du modèle NARMA de l'amplificateur.

3.7.4 Validation

Afin de valider l'efficacité de la prédistorsion NARMA dans le cas de l'amplificateur reconfigurable, le modèle illustré à la Figure 3.8 a été employé. Ainsi, un nombre important de vecteurs se situant principalement dans la zone de forte linéarité ont été générés de façon

aléatoire et soumis à l'amplificateur, précédé du module de prédistorsion. Les résultats de cette simulation sont visibles à la Figure 3.7.

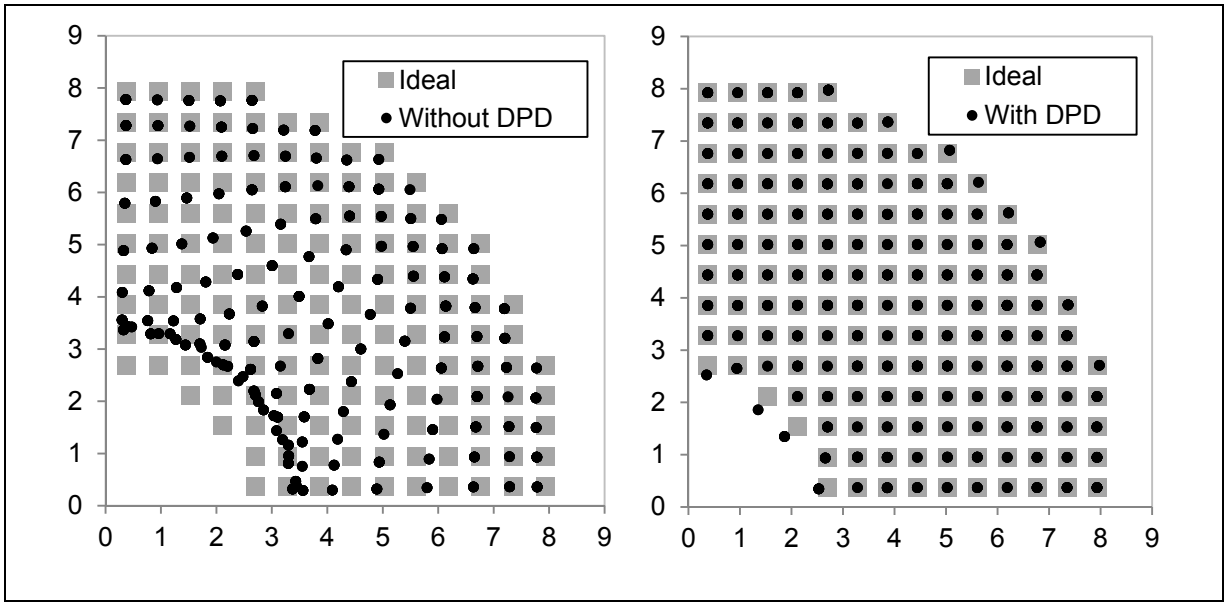


Figure 3.7 Vecteurs IQ idéaux et obtenus sans (gauche) et avec (droite) prédistorsion

On observe que sans prédistorsion, les conditions hardware lors de la reconfiguration de l'amplificateur provoquent de très fortes variations en amplitude et en phase, ce qui rend les symboles dans la constellation IQ pratiquement inutilisables.

Avec la prédistorsion activée, la plupart des symboles retrouvent leur position idéale. On note toutefois que certains symboles aux valeurs de tension minimales et maximales sont décalés. Ceci s'explique probablement par la modélisation de l'amplificateur qui n'est pas aussi bonne aux extrémités en raison de la plus faible densité d'échantillons dans ces zones.

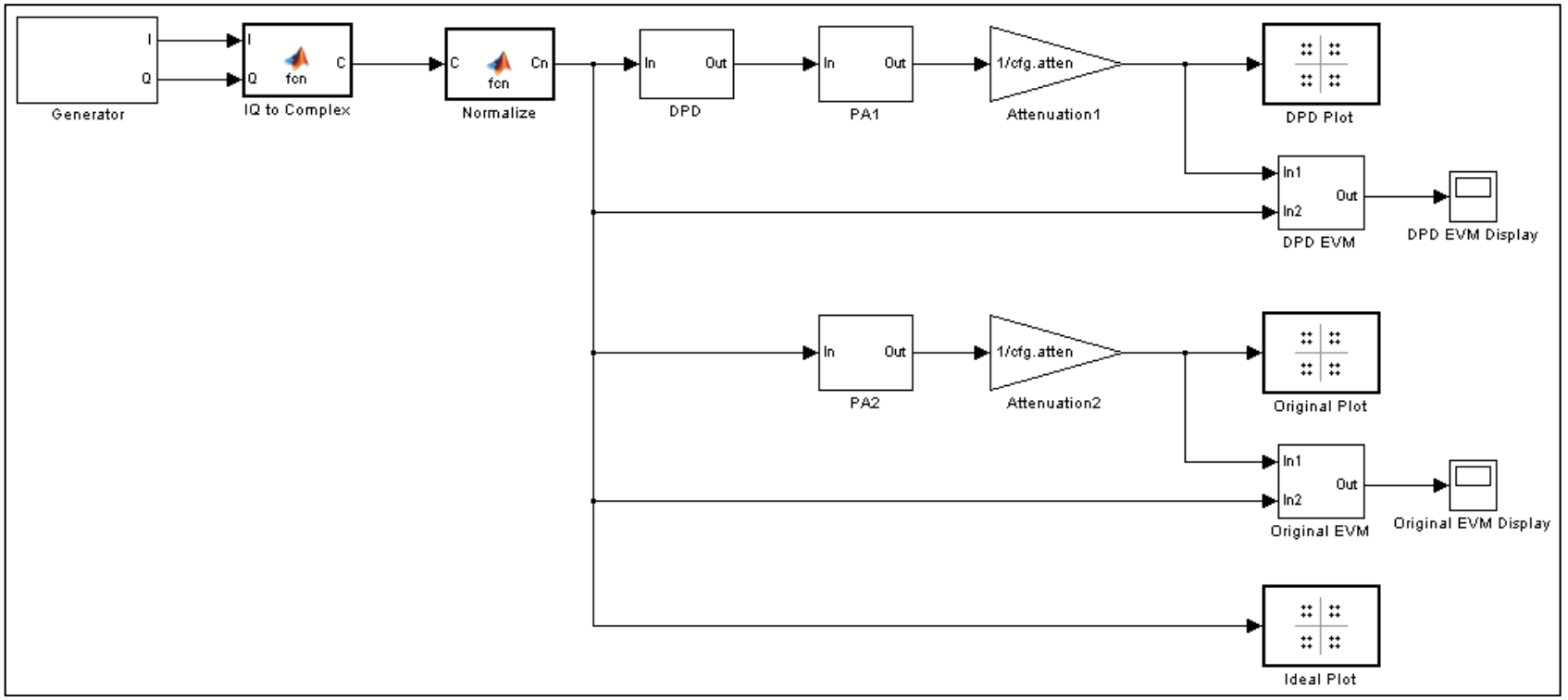


Figure 3.8 Modèle Simulink permettant de valider l'efficacité de la prédistorsion

3.8 Conclusion

Dans ce chapitre, nous avons vu les caractéristiques d'une architecture d'amplificateur reconfigurable dont la réponse AM-AM est non monotone. Nous avons vu en quoi la non monotonie pose problème dans l'application d'une méthode de prédistorsion de type NARMA. Le caractère insuffisant des techniques standard pour traiter ce genre de conditions a été mis en lumière, et une nouvelle approche sous forme d'une équation explicite, basée sur une analyse mathématique de l'interaction de f_0 et f_0^{-1}, a été proposée et expliquée. L'approche a été appliquée à une simulation de l'amplificateur dans MATLAB et Simulink. Les résultats permettent d'observer une amélioration de l'erreur vectorielle dans la région de réponse non monotone où l'amplificateur subit la reconfiguration hardware.

CHAPITRE 4

SYSTÈME DE PRÉDISTORSION PARAMÉTRABLE

4.1 Introduction

Ce chapitre traite du système de prédistorsion conçu et partiellement réalisé dans le cadre de cette maîtrise. Le but de ce système est de permettre de générer des signaux en bande de base et de les prédistorsionner. Afin de pouvoir tester différents scénarios de transmission, le système a été conçu de façon à être facilement paramétrable. Le système, illustré à la Figure 4.1, est constitué de trois parties principales. D'abord, un logiciel de paramétrisation sur PC permet de modifier les paramètres du système via une interface Ethernet. Ensuite, le système de contrôle s'exécutant sur le processeur Microblaze, instancié sur la puce FPGA du PicoSDR, traite les instructions reçues. Finalement, ce système synchronise les instructions avec l'horloge du système et assigne les registres et mémoires qui contrôlent les blocs de la chaîne de génération de signaux. Les blocs se trouvent donc immédiatement modifiés. Un des grands avantages, qui est aussi la plus grande difficulté de ce système, c'est la flexibilité associée à la configuration à partir du PC. De façon classique, une modification des blocs de la chaîne de transmission nécessiterait de procéder à une nouvelle synthèse du code VHDL, ce qui entraîne des délais importants, et constitue souvent un casse-tête pour l'utilisateur. Dans le cas présent, le système évite cela en utilisant des registres et mémoires routés jusqu'au PC, et modifiables en tout temps.

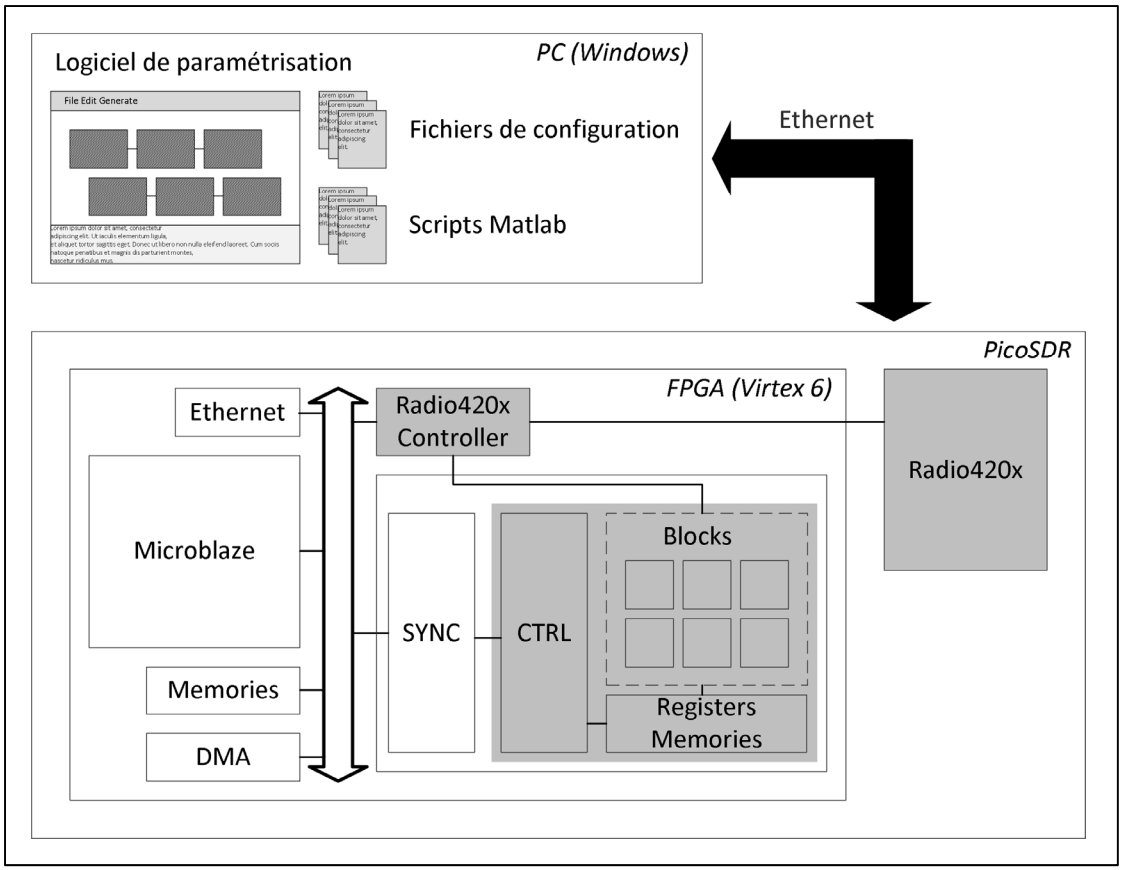


Figure 4.1 Schéma global du système de prédistorsion

Dans les sections qui suivent, nous verrons d'abord le logiciel de paramétrisation qui est utilisé pour configurer le système de génération. Ensuite, nous verrons le système de contrôle qui s'exécute sur le PicoSDR et qui agit à titre d'interface entre le lien Ethernet et les blocs de la chaîne de transmission. Finalement, nous verrons certains des blocs les plus importants de cette chaîne.

4.2 Logiciel de paramétrisation

Le logiciel de paramétrisation permet à l'utilisateur de commander le système. Son interface principale correspond à l'architecture du système de transmission. L'utilisateur peut donc sélectionner les blocs du système de transmission et modifier leur configuration. Pour une productivité maximale, le logiciel peut appeler des scripts MATLAB pour générer les

données de configuration. Le logiciel permet aussi de stocker les paramètres au format XML pour une utilisation ultérieure. Enfin, une fois les paramètres configurés, il communique avec le système de contrôle en lui envoyant les paramètres désirés. Dans les sous-sections suivantes, nous nous concentrerons sur le fonctionnement du logiciel et des choix technologiques qui ont été faits.

4.2.1 Technologie d'affichage

Il existe une multitude de langages de programmation et de technologies d'affichage. Sous Windows, certaines des plus communes sont Windows Forms et WPF, généralement codées en C# ou VB.NET, QT en C++ et Swing en Java. Le choix de la bonne technologie revêt une importance capitale afin d'assurer une productivité optimale ainsi qu'une bonne capacité de collaboration avec d'autres librairies et outils. Dans une vision à plus long terme, la facilité de modification et de maintenance, sont aussi des facteurs importants.

La solution retenue a été WPF en C# étant donné que le logiciel s'exécute sous Windows. Or, WPF est présentement la technologie privilégiée par Microsoft. WPF fait partie du cadriciel .NET, un ensemble de librairies puissantes qui permettent de faciliter la réalisation des nombreuses autres tâches que le logiciel doit accomplir. Avec WPF, l'affichage est fortement découplé des données. Les fenêtres ou contrôles sont principalement codés en XAML, un dialecte XML. C'est dans un fichier XAML que la hiérarchie des contrôles utilisateurs est définie. Il existe des contrôles permettant d'organiser d'autres contrôles, et des contrôles permettant d'afficher ou de modifier des données.

Un des avantages de WPF, c'est la possibilité de lier l'interface graphique avec le modèle de données grâce à des mécanismes de liaison de données (Voir ANNEXE V, p.99). Lorsque les données changent, des événements sont lancés et sont captés par les mécanismes en question. Ceux-ci mettent ensuite à jour l'interface graphique. Ils peuvent aussi procéder à l'inverse : à partir des changements de l'interface graphique – par exemple, lorsque l'utilisateur modifie les données – le modèle de données est mis à jour automatiquement. Le programmeur a donc

très peu à faire pour appliquer ou afficher les changements dans les données. Aussi, la réponse aux opérations de l'utilisateur est facilitée notamment par l'utilisation des différents mécanismes des classes Trigger, DataTrigger ou EventTrigger (Voir ANNEXE VI, p.101) qui permettent d'exécuter des commandes en fonction des changements au niveau des données ou des actions de l'utilisateur.

4.2.2 Interaction avec MATLAB

Plusieurs des blocs internes du système de transmission sont paramétrables avec plusieurs centaines, voire plusieurs milliers de données. Pour l'essentiel, il s'agit de coefficients de filtres numériques ou de tables de correspondance permettant de générer des signaux. Il est impensable que l'utilisateur entre ces coefficients manuellement. Un logiciel commun pour calculer de telles données est MATLAB.

Initialement, l'approche envisagée pour transférer les données vers le logiciel de paramétrisation a été d'utiliser des fichiers de données qui seraient générés par des scripts exécutés directement dans MATLAB par l'utilisateur. Cette approche comporte toutefois de nombreux désavantages. Le plus important, c'est la nécessité d'établir un format de fichier standard. L'utilisateur serait alors devenu responsable de respecter le protocole d'écriture, ce qui peut facilement devenir complexe en fonction des différents modules du système, et de la taille variable de certaines données. Un autre point négatif de cette approche est le manque d'intégration entre les scripts et le logiciel : on peut facilement perdre la trace des scripts initiaux, ou des paramètres qui ont été utilisés pour appeler ces scripts. Les deux systèmes étant complètement découplés, l'utilisateur doit aussi connaître exactement dans quel ordre procéder.

Afin de répondre à ces inconvénients, une seconde approche a été évaluée. Celle-ci consistait à laisser l'application générer les données. Il aurait donc fallu implémenter des algorithmes standards dans l'application. L'utilisateur aurait donc sélectionné un des algorithmes dans l'interface du logiciel, et entré les paramètres nécessaires. L'application aurait donc été

responsable de stocker les paramètres et les résultats. Le plus grand désavantage de cette approche est la nécessité de recompiler l'application afin de permettre à l'utilisateur de nouvelles façons de calculer les données. Il peut s'avérer difficile de programmer en C# avec WPF sans expérience préalable. Il faut donc compter un temps non négligeable pour effectuer de telles modifications.

La solution retenue consiste plutôt à interfacer l'application directement avec le logiciel MATLAB. Autrement dit, plutôt que d'exécuter les calculs dans l'application, on utilise MATLAB à cet effet. L'utilisateur n'a donc seulement qu'à coder ses scripts MATLAB et l'application se charge de les invoquer correctement. Il existe plusieurs façons d'interfacer une application avec MATLAB. En ce qui concerne l'invocation, la façon la plus simple consiste à utiliser la ligne de commande. Bien que cela convienne pour les fonctions retournant des données très simples comme un scalaire, il est toutefois difficile d'obtenir des résultats complexes comme des matrices, car il faut les déduire à partir des chaînes de caractères générées par MATLAB. Les deux autres possibilités envisagées sont l'utilisation de MATLAB .NET Builder et MATLAB Compiler. Alors que le premier génère du code .NET à partir d'un script MATLAB, le second génère du code C++. Dans un cas comme dans l'autre, la difficulté est qu'en cas d'une modification du script, les librairies générées doivent être recompilées et l'application doit donc l'être aussi.

Afin de pallier aux différents problèmes énoncés précédemment, un lien COM a plutôt été établi entre le logiciel et MATLAB. Il n'est pas nécessaire de recompiler l'application lorsque les scripts sont modifiés. Dans le logiciel, un contrat est établi en ce qui concerne les paramètres de sortie d'une fonction MATLAB. L'utilisateur sélectionne donc un script MATLAB fourni ou un autre codé par lui-même, et entre les paramètres qu'il veut lui fournir. Le logiciel stocke la référence au script et les paramètres. Par la suite, le logiciel invoque MATLAB au moyen du lien COM. Après l'exécution d'un script global, il invoque la fonction de l'utilisateur. Finalement, il lit les résultats via son lien COM, en se basant sur le contrat qu'il a défini, et les stocke en mémoire. Le Tableau 4.1 résume les différentes approches envisagées et leurs principaux avantages et inconvénients.

Tableau 4.1Avantages et inconvénients des approches évaluées pour calculer les données de configuration

Approche	Avantages	Inconvénients
Manuelle	Indépendance face au logiciel mathématique	Entrée fastidieuse et manque d'intégration
Fichiers standards	Indépendance face au logiciel mathématique	Responsabilisation forte de l'utilisateur et manque d'intégration
Fonctions mathématiques intégrées	Non nécessité d'une application tierce et simplicité d'utilisation	Manque de flexibilité et complexité de modification
Invocation de MATLAB en ligne de commande	Facilité d'invocation	Complexité de récupération des résultats
MATLAB .NET Builder ou MATLAB Compiler	Obtention directe des résultats	Nécessité de recompiler l'application lors de la modification des scripts MATLAB
COM	Obtention directe des résultats.	Nécessite un protocole pour l'organisation des résultats en sortie du script

La Figure 4.2 contient la fenêtre de configuration du script MATLAB qui calcule les coefficients du filtre RRC. Vis-à-vis de « Return », l'application décrit ce qu'elle attend en retour du script. Dans ce cas, il s'agit d'un vecteur de 129 coefficients. Dans le premier champ, l'utilisateur sélectionne le script à exécuter afin de générer les données. Il peut s'agir d'un script fourni par défaut, ou d'un script personnalisé. Une fois la sélection complétée, le contenu du script est affiché. En cliquant sur « Edit », il peut être modifié dans Notepad. Dans le champ « Arguments », l'utilisateur entre les paramètres pertinents à l'utilisation du script.

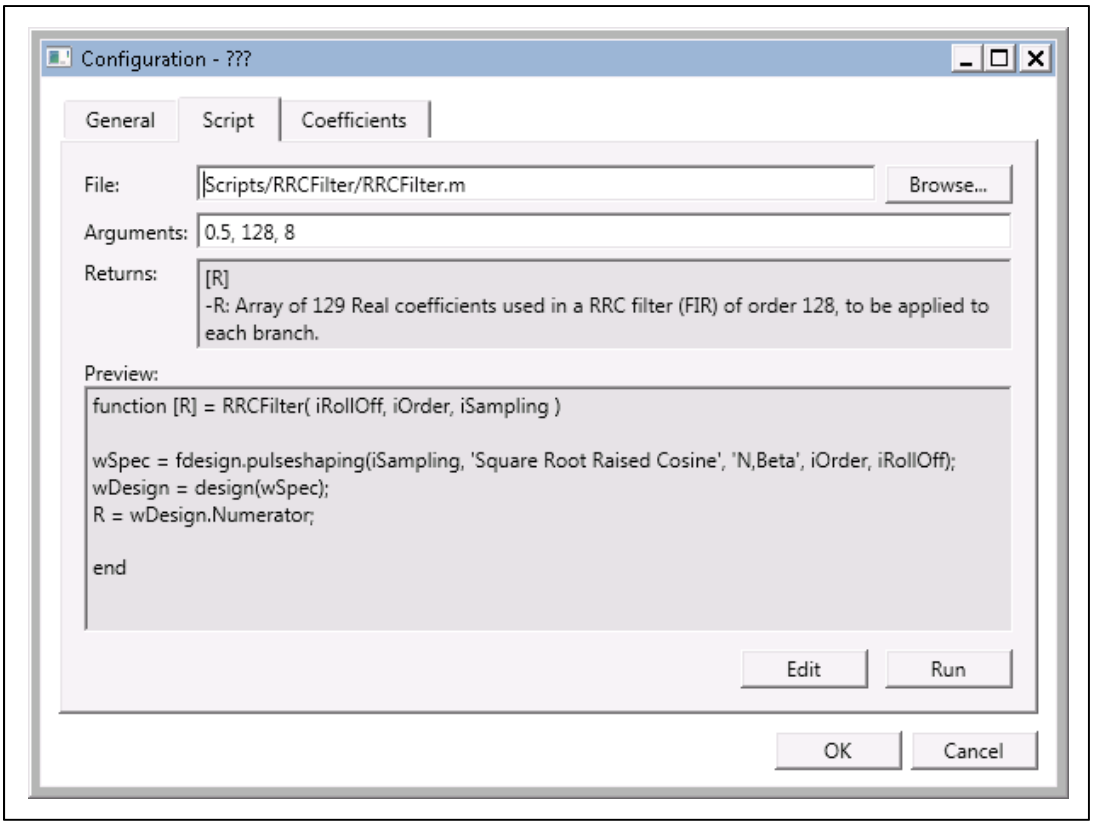


Figure 4.2 Configuration du script MATLAB du filtre RRC

Lorsque l'utilisateur clique sur « Run », la boîte de dialogue illustrée à la Figure 4.3 indique qu'une opération est en cours d'exécution. Un observateur d'événements permet de visualiser les différentes étapes et avertit l'utilisateur en cas de problème. Lors de l'exécution, l'application commence par exécuter un script MATLAB global. Ce script permet entre autres de définir des variables globales pouvant être réutilisées dans les autres scripts. Enfin, le script du module est appelé avec les paramètres définis par l'utilisateur. Lorsque les résultats sont retournés, l'application va chercher les informations attendues selon la convention définie.

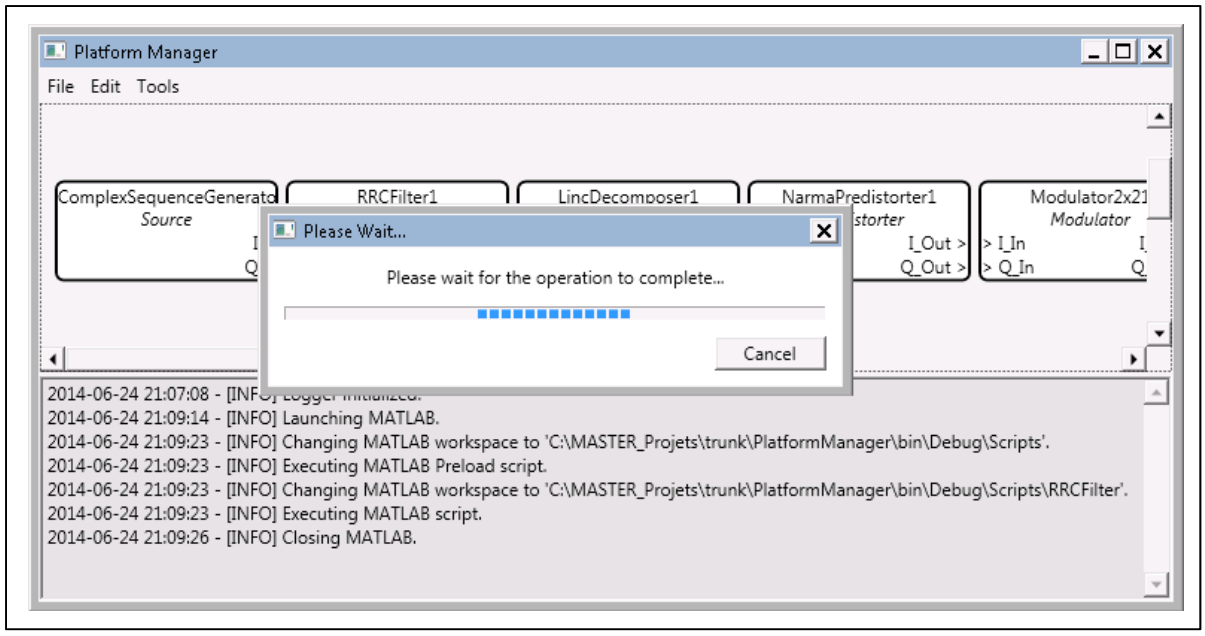


Figure 4.3 Exécution d'un script MATLAB

4.2.3 Sauvegarde et restauration XML

Afin de permettre à l'utilisateur d'interrompre une session de travail et de reprendre plus tard, il est essentiel de pouvoir sauvegarder l'état de la configuration du système. Il existe plusieurs façons de réaliser la sauvegarde d'un ensemble de données. On nomme sérialisation le processus de conversion de l'état d'un modèle vers un fichier ou un flux de données. La désérialisation quant à elle, effectue l'opération inverse qui consiste à restaurer l'état de l'application.

La sérialisation binaire est une des méthodes les plus efficaces car elle consiste à stocker presque directement le contenu de la mémoire du programme. Ce contenu peut donc être écrit et lu très rapidement, ce qui en fait le choix par excellence dans un contexte où la performance joue un rôle de premier plan, ce qui n'est pas le cas ici. Aussi, ce format est très difficile à comprendre pour un humain, et sa modification l'est tout autant. Advenant une modification de la taille ou de l'organisation des données dans une version subséquente de

l'application, et à moins qu'un mécanisme n'ait été prévu, il est fort probable que les fichiers générés par les versions antérieures deviennent inutilisables.

La sérialisation de type linéaire dans un fichier de configuration « ini » consiste à stocker l'information en format texte, avec peu ou pas de structure. L'avantage principal de cette approche est la facilité de compréhension et de modification. Toutefois, son manque de structure peut poser des problèmes de performances et rend difficile la représentation d'une architecture logicielle complexe.

La sérialisation au format XML répond au problème de structure du format précédent. Il permet de hiérarchiser l'information en un arbre de données constitué de nœuds définis récursivement, auxquels on peut définir des attributs et un contenu. La librairie .NET permet de traiter ces fichiers de façon très efficace au moyen d'outils puissants et très bien optimisés. On navigue dans un fichier XML au moyen de requêtes Xpath. L'insertion ou la recherche de contenu est très facile, ce qui fait du format XML un choix idéal pour la sérialisation, bien qu'il soit néanmoins plus lent que le format binaire. Notons toutefois qu'il faudra plus d'espace sur disque pour stocker les informations en raison des métadonnées.

La librairie .NET permet de sérialiser les données d'une application au format XML. Afin d'utiliser cette fonctionnalité, il suffit de spécifier l'attribut « Serializable » sur les classes. Par défaut, l'outil de sérialisation/désérialisation procède de façon linéaire à l'intérieur d'une classe. Bien que ce soit utile pour un système dont l'architecture logicielle est figée, cette façon de procéder s'est rapidement révélée peu pratique dans le cas d'une application en cours de développement. En effet, les changements fréquents du logiciel avaient pour effet de rendre les anciens fichiers de sauvegarde inutilisables. Afin de pallier à ce problème, une méthode de sérialisation/désérialisation XML sur mesure a été implémentée. Elle consiste à procéder à une recherche plutôt que de procéder selon un ordre particulier. D'une part, lorsqu'un objet ou un attribut inconnu est présent dans le fichier, il sera simplement ignoré. D'autre part, dans le cas où, au contraire, un objet ou un attribut attendu est manquant, la recherche Xpath ne retourne rien et une valeur par défaut est utilisée. Les désavantages de

procéder de cette façon sont la nécessité de charger le fichier complet en mémoire et le temps de traitement additionnel requis par les recherches Xpath. Néanmoins, ces désavantages n'ont pas d'impact significatif dans le cas qui nous concerne en raison de la quantité limitée de données de cette application.

L'implémentation du processus de sérialisation décrit au paragraphe précédent repose sur deux classes principales. Pour les types simples comme les entiers ou des nombres à virgule flottante, le mécanisme des TypeConverter de la librairie .NET est employé. Il assure la sérialisation en convertissant ces valeurs en chaînes de caractères, et permet la désérialisation en effectuant l'opération inverse. Les types simples sont stockés en tant qu'attributs de l'élément XML de la classe qui les contient. Pour les types complexes, qui souvent, contiennent plusieurs données, l'interface IFlexibleXmlSerializable a été définie (Voir ANNEXE VII, p.103). Lors de la sérialisation, un objet d'une classe qui implémente cette interface est appelé par son parent. Il reçoit son nœud XML de sérialisation en paramètre. Il doit donc stocker ses attributs simples et complexes. Pour les objets de types complexes, il crée un nœud XML et utilise l'interface IFlexibleXmlSerializable pour assurer la sérialisation de ces objets de façon récursive. Enfin, l'objet appelle sa classe de base si nécessaire, afin de compléter sa sérialisation. Les opérations de sérialisation sont grandement facilitées par l'utilisation de la classe utilitaire XmlSerializationHelper (Voir ANNEXE VIII, p.105) qui permet de réaliser les différentes étapes en un minimum de lignes de code. La désérialisation s'effectue de façon équivalente et elle est aussi grandement facilitée grâce un outil : il s'agit cette fois de la classe utilitaire XmlDeserializationHelper (Voir ANNEXE IX, p.109).

Un exemple complet de sérialisation est le cas de la classe RRCFilter (Voir ANNEXE X, p.113). Lors de la sérialisation, la première opération consiste à sérialiser sa classe de base, Block2x2 (Voir ANNEXE XI, p.115). La sérialisation remonte jusqu'à la classe racine qui implémente l'interface IFlexibleXmlSerializable, Block (Voir ANNEXE XII, p.117). Cette dernière sérialise trois variables de types simples. L'exécution retourne ensuite avec Block2x2 qui sérialise quatre types complexes. Finalement, RRCFilter termine avec deux variables de types simples et une collection de types simples. L'exécution retourne donc à

son parent qui continue le processus jusqu'à l'obtention du fichier XML complet (Voir ANNEXE XIII, p.119).

4.3 Système de contrôle

Dans la section précédente, nous avons vu le logiciel de paramétrisation qui s'exécute sur un PC. Cette section-ci porte plutôt sur une partie du système qui s'exécute sur la puce FPGA du PicoSDR et pour laquelle on peut se référer à la Figure 4.1. Le système de contrôle a pour rôle de recevoir les commandes en provenance de l'interface Ethernet et de contrôler les blocs internes en conséquence. Il peut aussi lire des données sur les blocs internes et les transmettre via l'interface Ethernet. Le système de contrôle est constitué d'un processeur Microblaze instancié, d'une interface Ethernet, d'un accès mémoire et d'une interface au bus principal sur lequel est raccordé le cœur de la chaîne de transmission, le module PSCore. Finalement, il comprend une couche de synchronisation dans PSCore et les registres et mémoires qui servent à modifier le fonctionnement des blocs internes.

4.3.1 Communications TCP/IP

La communication entre le logiciel de paramétrisation et le système de contrôle est conçue pour être faite via un lien de type Gigabit Ethernet. Selon les spécifications du PicoSDR, il est possible d'atteindre jusqu'à 900 Mbps de façon soutenue avec ce lien. Il aurait été possible de réaliser la communication avec un lien PCI Express 4x pour atteindre 6.4 Gbps. Néanmoins, une telle bande passante excède de loin les besoins du système pour transmettre en continu les données reçues ou à transmettre. En effet, pour une lecture en continu sur deux canaux de 12 bits à une cadence de 14 MSPS, nous arrivons, en excluant la surenchère due au protocole de communication, à 336 Mbps ce qui est nettement sous la spécification du lien Gigabit Ethernet. On arrive à ce résultat en utilisant l'équation suivante, où T est le taux binaire, B le nombre de bits par symbole, C le nombre de canaux et F la fréquence maximale.

$$T = BCF \tag{4.1}$$

Les variables B, C et F dépendent des spécifications de la plateforme. Tandis que la signification des variables B et C est plus intuitive, on ne peut en dire autant du choix de F. Pour cette dernière, on fait une surestimation basée sur le critère de Nyquist. Puisque la bande passante est d'au plus 28 MHz, un signal ayant une fréquence d'au plus 14 MHz peut théoriquement être utilisé. Dans un tel cas, il s'agit d'une sinusoïde idéale dont l'amplitude ou la phase ne varient pas. En réalité toutefois, ces deux paramètres varient. Afin de pouvoir reconstruire le signal au récepteur, le transmetteur devra donc transmettre des signaux qui varient à une fréquence de moins de 14 MHz.

Le protocole bas niveau utilisé pour la communication entre le système de contrôle et le PC est TCP/IP. Il aurait été possible de procéder directement avec la couche MAC, via des outils tels que Pcap.Net. Néanmoins, il aurait alors fallu implémenter manuellement les moyens d'assurer la qualité des données reçues, ainsi qu'un mécanisme de retransmission. Aussi, cela aurait ajouté la limitation d'avoir un lien local entre le PC et la plateforme. Pour ces raisons, la couche de transport de données utilisée est TCP de la suite TCP/IP. De cette façon, la fiabilité de la transmission/réception est assurée, tout en permettant que la plateforme soit contrôlée à partir de l'extérieur de son réseau local. Le logiciel de paramétrisation peut facilement communiquer avec des sockets TCP/IP grâce à la librairie .NET qui fournit une implémentation complète. Pour ce qui est du système de contrôle par contre, il n'y a pas de solution aussi simple. Toutefois, il est possible d'utiliser la librairie lwIP pour obtenir une fonctionnalité similaire.

Une fois le protocole bas niveau défini, il reste à définir quelles données seront échangées entre le logiciel de paramétrisation et le système de contrôle. Il s'agit donc de définir le protocole haut niveau qui permet de préparer et d'interpréter les données échangées. Un entête standard a été défini afin de baliser l'ensemble des communications de façon simple. À la Figure 4.4, on peut voir qu'un message commence toujours par une alternance de 1 et 0 sur 32 bits. Ensuite, le type de message est indiqué. Les deux premiers bits de ce champ indiquent s'il s'agit d'une requête, d'un ping, d'un accusé de réception, ou d'une demande de réinitialisation. Lorsqu'il s'agit d'une requête, le bit 2 indique s'il s'agit d'une lecture ou

d'une écriture, tandis que le bit 3 indique s'il s'agit d'une lecture simple ou en cascade. Finalement, pour les écritures, le bit 4 permet de spécifier si on doit masquer les données écrites afin d'écrire certains bits en particulier.

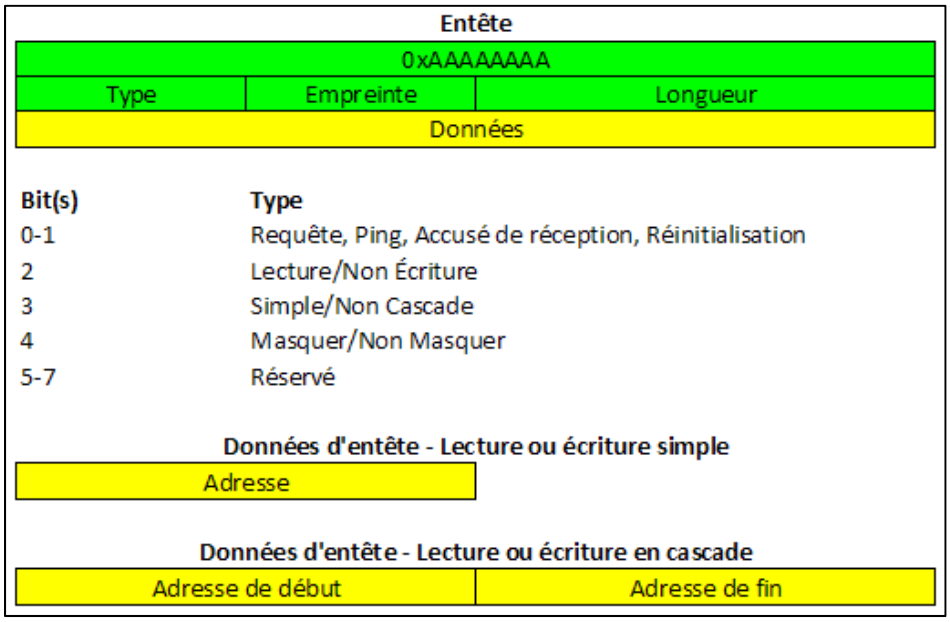


Figure 4.4 Éléments de l'entête

Le champ suivant est l'empreinte du message, qui est simplement une opération XOR entre ses octets, en considérant ce champ à zéro. Le dernier champ est la longueur du message en octets. Finalement, on retrouve les données de l'entête, qui sont une adresse de lecture/écriture simple, ou d'une plage d'adresses pour une lecture/écriture en cascade. Une fois une requête reçue, un accusé de réception est envoyé par le système de contrôle, et les données sont ensuite transférées dans un sens ou dans l'autre selon qu'il s'agisse d'une lecture ou d'une écriture. Dans le cas d'un message de type Ping, le système de contrôle doit simplement accuser réception dans les plus brefs délais afin de confirmer son fonctionnement au logiciel de paramétrisation : aucun transfert de données subséquent n'est alors attendu.

4.3.2 Adressage

Un système d'adressage particulier a été défini afin de pouvoir contrôler les différents blocs du système. Afin d'éviter de devoir faire du cas par cas pour chaque bloc, l'espace d'adressage a été découpé en différents niveaux. Ces niveaux sont ensuite assignés aux différents blocs selon leurs besoins. Tout d'abord, il faut savoir qu'une adresse réfère à un bloc de mémoire de quatre octets, et non à un octet comme on voit traditionnellement pour les mémoires. On peut donc considérer que chaque case mémoire a une largeur de 32 bits. Ce choix a été fait afin de simplifier le protocole d'écriture qui demanderait de supporter des lectures selon un alignement de 1, 2 ou 4 octets. La plage d'adresses va de 0x00000000 à 0x01000000 à partir de l'adresse de base du périphérique, pour un total de 16777216 valeurs de 32 bits adressables. Le Tableau 4.2 montre comment les plages ont été découpées. En tout on retrouve 256 registres, et trois sections de 240 mémoires pouvant avoir 256, 4096 ou 65536 valeurs.

Tableau 4.2 Plages d'adressage

Plage	Min	Max	Description
A	0x00000000	0x000000FF	256 registres
B	0x00000100	0x00000FFF	Réservé
C	0x00001000	0x0000FFFF	240 mémoires de 256 valeurs
D	0x00010000	0x000FFFFF	240 mémoires de 4096 valeurs
E	0x00100000	0x00FFFFFF	240 mémoires de 65536 valeurs

Les plages ont été séparées de façon à permettre un décodage rapide des adresses sans nécessiter de comparaisons mathématiques comme plus-petit ou plus-grand. La signification des données d'une adresse est indiquée dans le Tableau 4.3. Ainsi, pour un registre (plage A), il suffit de vérifier que les octets #3, #2 et #1 soient nuls. Pour une mémoire 256 valeurs (plage C), il suffit de vérifier que les octets #2 et #3 soient nuls, et que l'octet #1 soit non nul. On procède de façon similaire pour les plages D et E. L'exception pour la plage D, est qu'on

considère séparément les quatre bits inférieurs et supérieurs de l'octet #2. L'absence d'opérations mathématiques telles que « plus petit » ou « plus grand » dans le mécanisme de décodage permet d'effectuer le décodage rapidement en minimisant la surface nécessaire.

Tableau 4.3 Signification des données d'adressage

<table>
<tr><th>Plage</th><th>Octet 3</th><th>Octet 2</th><th>Octet 1</th><th>Octet 0</th></tr>
<tr><td>A</td><td></td><td></td><td></td><td>ID Registre</td></tr>
<tr><td>B</td><td></td><td></td><td></td><td></td></tr>
<tr><td>C</td><td></td><td></td><td>ID mémoire</td><td>Addr. mémoire</td></tr>
<tr><td>D</td><td></td><td></td><td>ID mémoire</td><td>Addr. mémoire</td></tr>
<tr><td>E</td><td></td><td>ID mémoire</td><td colspan="2">Addr. mémoire</td></tr>
</table>

Afin de simplifier la compréhension du routage, les adresses utilisées lors des communications TCP/IP entre le logiciel de paramétrisation et le système de contrôle sont les mêmes qui sont utilisées sur le bus de données du Microblaze, à l'exception de l'ajout d'une adresse de base dans ce dernier cas.

4.3.3 Synchronisation des horloges

Une des particularités du fonctionnement du Radio420x est qu'il génère lui-même son horloge. Le système de contrôle assigne les registres qui déterminent le fonctionnement du générateur d'horloge. Cette horloge est ensuite utilisée pour obtenir les valeurs I/Q à transmettre, ou à les récupérer de la chaîne de réception. Le système de contrôle, par contre, fonctionne sur une horloge différente dont la fréquence est fixée lors de la synthèse. Nous nous retrouvons donc avec deux domaines d'horloge complètement indépendants, tant en fréquence qu'en phase.

Il n'est pas trivial de communiquer adéquatement entre deux domaines d'horloges différents, principalement en raison de la métastabilité. Il s'agit d'un sujet complexe qui dépasse

largement les considérations de ce projet de recherche. Sans entrer dans les détails, il convient toutefois d'expliquer la nature de ce problème important. Le code VHDL comportemental est principalement basé sur l'utilisation de bascules synchrones (flip-flops en anglais) de type D. Une telle bascule est un élément de mémoire qui reçoit un signal d'horloge, une entrée et possède une sortie. La donnée à l'entrée est stockée au front montant de l'horloge, et devient alors disponible en sortie. Afin que le processus d'acquisition fonctionne bien, il est essentiel que le signal à l'entrée de la bascule respecte certaines contraintes : il doit être stable un certain temps avant l'arrivée du front montant de l'horloge et le rester un certain temps après ce dernier. Autrement, la bascule tombe en état de métastabilité et sa sortie peut devenir soit 0 soit 1 après un certain temps.

En soi, ce problème peut très bien se produire avec un seul domaine d'horloge. On l'observe d'ailleurs lorsque la distance entre deux bascules est trop grande : le délai de propagation est tel que le signal ne s'est pas stabilisé assez tôt à l'entrée de la bascule, ce qui provoque un problème de métastabilité. Lorsque deux bascules sont reliées en cascade, et contrôlées par deux horloges indépendantes, ce n'est pas nécessairement la distance qui pose problème, mais plutôt l'asynchronisme. Pour ce qui est de la distance, il s'agit simplement de considérer l'horloge ayant la fréquence la plus élevée pour contraindre le synthétiseur.

Dans la plupart des cas de figure, la solution la plus simple pour régler le problème de synchronisation consiste à utiliser une structure de type FIFO. L'outil CORE Generator System de Xilinx permet de générer une telle structure automatiquement. On peut déterminer la profondeur de la file, la largeur des données, ainsi qu'une panoplie d'autres options. Initialement, c'est l'implémentation qui a été privilégiée en raison de la fiabilité de l'implémentation résultante, et de la simplicité de modification des différents paramètres. Néanmoins, une simulation a permis de découvrir qu'une latence d'environ sept cycles était nécessaire entre l'écriture et la lecture. Une telle latence est peu problématique lorsqu'il est simplement question de transférer de nombreuses données en cascade. Le problème est tout autre lorsqu'il s'agit d'encapsuler une interface de lecture/écriture. En effet, il ne faut pas seulement une FIFO, mais bien deux. Le maître peut donc envoyer ses requêtes par une des

deux FIFO, et l'esclave répond par l'autre. Dans le cas d'une opération de lecture, il faut donc compter sept cycles afin de donner l'ordre de lecture, au moins un cycle afin d'obtenir le résultat, et sept cycles additionnels pour transférer la donnée du côté du maître. Le problème est amplifié lorsque l'esclave a une fréquence d'horloge plus faible. Si par exemple, sa fréquence d'horloge est deux fois plus basse que celle du maître, ses cycles d'exécution prendront deux fois plus de temps du point de vue de ce dernier. Un délai de 15 cycles, peut donc facilement devenir l'équivalent de 23 cycles. La lecture d'une donnée peut donc provoquer un ralentissement du processeur qui peut compromettre d'autres opérations comme le traitement des communications TCP/IP, ou rendre difficile le transfert de données en temps réel.

Afin de résoudre ce problème, le mécanisme de synchronisation schématisé à la Figure 4.5, a été implémenté. Il est basé sur le mécanisme de synchronisation de données bidirectionnel suggéré par (Ginosar, 2011).

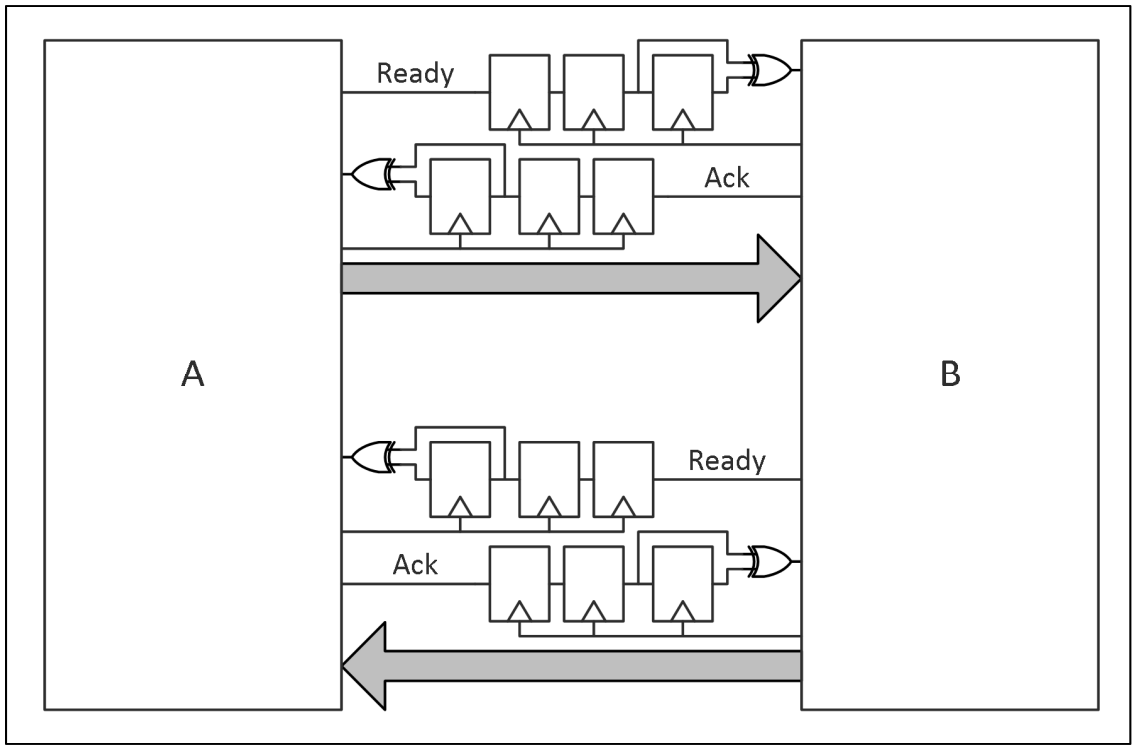


Figure 4.5 Synchronisation bidirectionnelle

Le module de synchronisation de base est constitué d'un bus de données, d'un signal servant à indiquer que ce dernier est prêt, et d'un signal servant d'accusé de réception. Afin d'effectuer une communication bidirectionnelle, deux unités sont nécessaires. En procédant ainsi, lorsque A transfère une instruction de lecture à B, B peut confirmer la réception de l'instruction sans qu'elle n'ait déjà été exécutée, ce qui permettra à A d'envoyer une seconde instruction sans attendre.

Une autre particularité de cette implémentation est l'ajout d'une troisième bascule et d'une porte logique de type XOR. Cela permet de détecter les changements de valeur des lignes Ready ou Ack. En l'absence d'un tel mécanisme, il faudrait réinitialiser les signaux à 0 avant de les changer à nouveau à 1. Le mécanisme implémenté permet donc d'effectuer une communication basée sur les changements. Par exemple, pour effectuer une transmission de A vers B, A assigne le bus de données, et inverse la valeur de la ligne Ready. Lorsque ce changement parvient à la sortie de la seconde bascule, la porte logique XOR retourne 1 pendant un cycle, ce qui active la lecture du module B. Le module B lit donc les données et inverse la valeur de la ligne Ack. Quand la porte XOR de cette ligne prend la valeur 1, A considère que B a reçu l'information et peut utiliser l'unité de synchronisation à nouveau. Dans le cas d'une instruction de lecture, B transmettra éventuellement une donnée via son unité de transmission en procédant de façon similaire à un transfert de A vers B. Évidemment, il ne s'agit que d'un exemple d'utilisation qui n'enlève rien au caractère générique de l'unité de synchronisation. C'est néanmoins le cas utilisé dans le système de contrôle, de façon à synchroniser les opérations de lecture et d'écriture en provenance du bus de données, vers les registres et mémoires utilisés par les blocs internes.

4.4 Blocs internes

Dans cette section, nous verrons certains des blocs internes du système. Les blocs internes ne sont pas en interaction directe avec le système de contrôle. Le système de contrôle manipule des registres ou mémoires, et ceux-ci sont ensuite utilisés par les blocs internes. Ils sont nombreux, et généralement construits d'une hiérarchie de plusieurs blocs. Vu leur nombre

important, nous verrons uniquement les détails de conception ou d'implémentation de certains des blocs les plus importants.

4.4.1 Générateur par table d'adressage

Le générateur par table d'adressage est un bloc qui sert à générer les données de bande de base du système. Il est essentiellement constitué d'une mémoire dynamique et d'un itérateur borné paramétrable. À sa sortie, on obtient deux valeurs de 16 bits, qui correspondent aux scalaires I et Q d'un vecteur. Dans le logiciel de paramétrisation, l'utilisateur peut définir manuellement les valeurs devant être transmises. Il peut aussi utiliser un script Matlab pour générer un signal, et l'échantillonner pour générer la table d'adressage. Une mémoire de type BRAM de 65536 valeurs de 32 bits (donc 65536 couples IQ) permet de conserver un nombre considérable de données pour une utilisation directe. Si on veut que le module transmette davantage de données, il est possible de segmenter la mémoire en deux parties. Ainsi, pendant que le module transmet un segment de la mémoire aux blocs suivants, le processeur ou un module DMA peut remplir les autres segments avec des données provenant d'un flux ou de la mémoire DRAM, qui elle, est beaucoup plus grande. Une fois cette étape terminée, l'itérateur peut transmettre l'autre section de la mémoire. Un autre mode de fonctionnement consiste à itérer continuellement sur une section contigüe de la mémoire. Les bornes d'itération sont stockées dans un registre modifiable par le système de contrôle.

4.4.2 Filtre RRC

Plutôt que de produire, à la fréquence de la bande de base, des changements brusques entre les valeurs IQ des différents symboles ce qui produirait une large bande passante, on filtre le signal avec un filtre de type RRC, dont la réponse impulsionnelle est illustrée à la Figure 4.6. Précédé d'une étape de suréchantillonnage, ce filtre est ensuite appliqué de façon à réduire l'interférence intersymbole du système. La limite de coefficients du filtre a été définie à 128, pour un total de 129 valeurs.

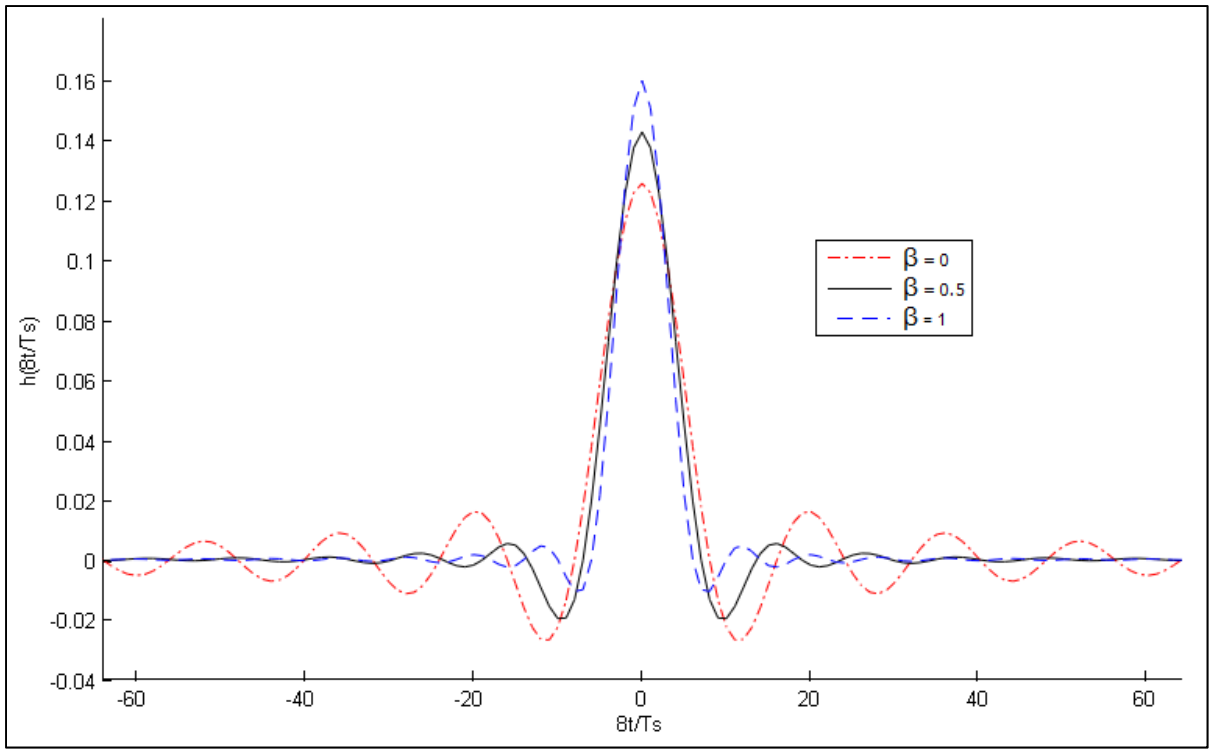


Figure 4.6 Réponse impulsionnelle d'un filtre RRC d'ordre 128 avec suréchantillonnage de 8 et différents facteurs β

La façon la plus simple d'implémenter l'additionneur de ce filtre est de procéder de façon séquentielle, tel qu'on voit à la Figure 4.7. Avec autant de valeurs à additionner, il est impensable d'effectuer le calcul en un seul cycle. Cela aurait pour impact de générer d'énormes délais de propagation, ce qui nécessiterait de réduire fortement la fréquence d'opération du système. Le plus grand problème de cette approche, c'est qu'il faut donc 129 cycles avant d'obtenir le résultat, et donc une donnée utilisable. Cela introduit une latence significative qui n'est pas acceptable en cas de boucles de rétroaction, ce qui est envisageable si la plateforme est éventuellement utilisée par la suite pour d'autres projets. Une façon plus rapide d'obtenir le résultat du calcul est d'utiliser un arbre d'additionneurs. Sa latence est le logarithme en base deux du nombre d'additions. Pour 129 coefficients, on obtient donc 8 cycles de latence, ce qui est une acceptable.

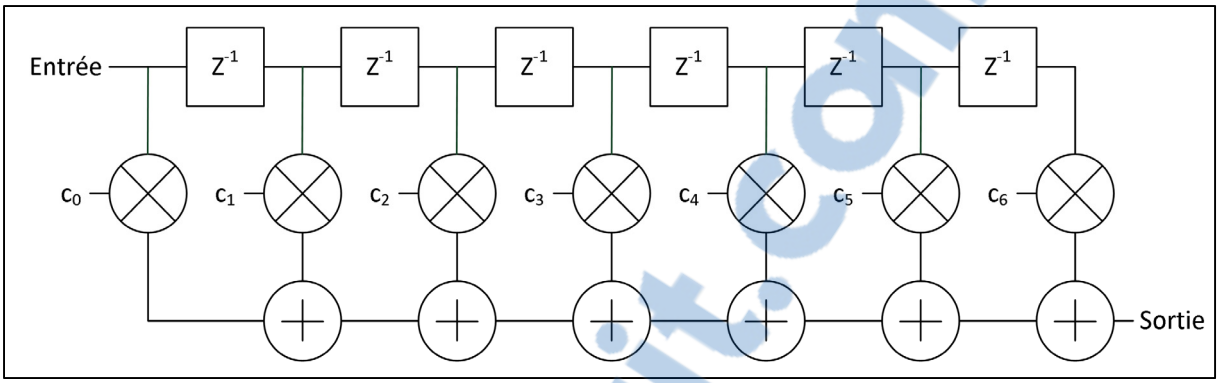


Figure 4.7 Addition séquentielle des produits

Afin de permettre une modification ultérieure facile du nombre de coefficients du filtre, l'additionneur a été codé de façon générique et récursive, au moyen d'instructions « generate » conditionnelles. Il ne suffit donc que de modifier un paramètre pour changer le nombre d'entrées. Ainsi, comme on le voit dans la Figure 4.8, pour un additionneur en arbre à 7 entrées, le premier niveau les reçoit et instancie 2 additionneurs : un de 4 données ainsi qu'un autre de 3 données, et il instancie aussi un additionneur simple pour la somme des résultats intermédiaires. Le même processus s'effectue récursivement pour l'additionneur du haut, car il s'agit d'une puissance de deux. Pour ce qui est de celui du bas, le niveau suivant est différent. Dans un premier temps, un additionneur à deux entrées est instancié, et un additionneur à une entrée, qui est remplacé par un fil. Toutefois, on détecte que le second additionneur n'a pas un nombre d'entrées du même ordre que le premier. En conséquence, un délai est ajouté. Ce délai permet donc d'assurer la cohérence des opérations.

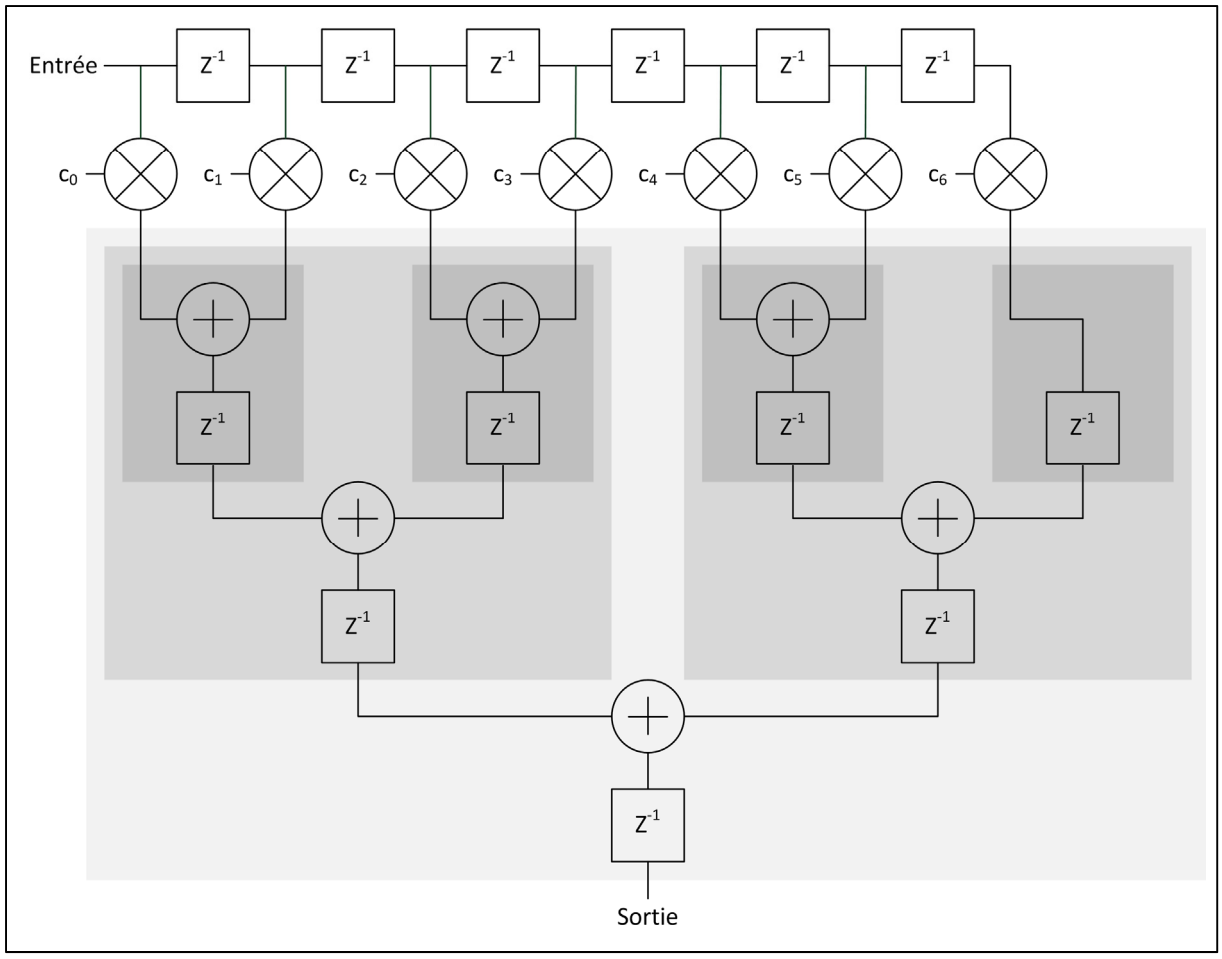


Figure 4.8 Addition des produits avec un additionneur en arbre récursif

À partir de la norme VHDL 2008 (IEEE, 2009), il est possible de définir des types génériques. Plutôt que d'avoir un bus commun pour les bits de toutes les entrées de l'additionneur, il aurait donc été possible de définir un array générique, qui aurait contenu une case pour chaque entrée. Cela aurait facilité la compréhension du code de l'additionneur, et l'aurait rendu encore plus simple à modifier. Cette approche n'a malheureusement pas pu être retenue car les outils de Nutaq nécessitent une version des outils de Xilinx à laquelle ceux-ci ne peuvent synthétiser que du code de la norme VHDL 1993 (IEEE, 1994).

En ce qui concerne les multiplicateurs, Xilinx Core Generator System a été utilisé pour les générer. On compte un multiplicateur pour chaque coefficient du filtre.

4.4.3 Module de prédistorsion

Le module de prédistorsion conçu est de type NARMA. Il est donc conçu en deux sections principales : la section à moyenne mobile (Moving-average), un filtre non récursif à réponse impulsionnelle finie, et la section autorégressive (Autoregressive), un filtre récursif à réponse impulsionnelle infinie. Ce qui rend le module non linéaire, c'est que dans un cas comme dans l'autre, les coefficients sont remplacés par des fonctions potentiellement non linéaires qui prennent un scalaire en entrée et retournent un nombre complexe en sortie. La façon la plus simple d'implémenter la fonction est d'utiliser une table de correspondance. Avec 2 paliers dans la section non récursive et 2 dans la section récursive, soit la configuration suggérée par (Gilabert *et al.*, 2008), il faut compter un total de 4 tables de correspondance. Afin d'éviter de dépasser les capacités de mémoire de la puce FPGA, ou de compromettre la possibilité d'utiliser la mémoire pour d'autres fins à l'avenir, la résolution en entrée a été limitée à 12 bits, ce qui implique que chaque table a une taille de 4096 entrées de deux valeurs de 16 bits, une pour la phase et une pour le module. Il convient donc d'assigner ces mémoires à des adresses dans la plage D.

Optionnellement, il serait possible d'implémenter une interpolation linéaire entre les cases adjacentes lors d'une consultation de la mémoire. Il faut alors prévoir une interface mémoire qui permet de lire deux valeurs en même temps pour éviter de devoir dupliquer les tables.

En ce qui a trait des additions, on compte jusqu'à 2 additions séquentielles à effectuer par cycle pour la section à réponse impulsionnelle finie. Bien que ce soit un nombre beaucoup plus raisonnable que les 129 additions nécessaires pour le filtre RRC, il s'agit quand même d'un nombre non négligeable si on envisage de les effectuer toutes en même temps. L'additionneur implémenté pour le filtre RRC étant bien modularisé, l'approche la plus simple est donc de réutiliser cette composante. Il représente la section ombragée de la Figure 4.9.

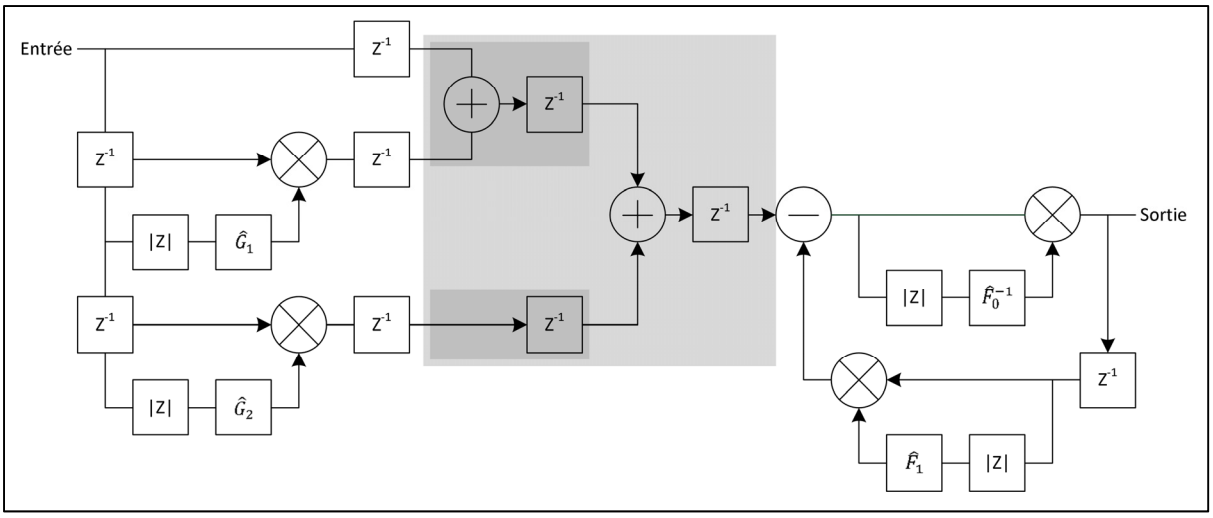


Figure 4.9 Architecture du module de prédistorsion

La dernière section avant la sortie est néanmoins celle qui est la plus exigeante en termes de délais de propagation. En effet, comme il s'agit d'un filtre à réponse impulsionnelle infinie, le pipelinage est difficile à réaliser, tel qu'expliqué dans (Francis, 2009), d'autant plus que le filtre est non linéaire.

4.4.4 Échantillonneurs

Les échantillonneurs permettent de collecter des données et de les renvoyer au logiciel de paramétrisation sur demande. Ils sont très utiles lors des phases d'implémentation puisqu'ils permettent de prendre des captures sur mesure des différents signaux. On peut les comparer aux périphériques servant au déverminage avec l'outil ChipScope de Xilinx. Contrairement à ces derniers, les échantillonneurs sont accédés via l'adressage standard par le système de contrôle et leur contenu peut être transféré au PC par l'interface Ethernet.

Le fonctionnement des échantillonneurs est très simple. L'activation, la désactivation ou la réinitialisation d'un échantillonneur sont contrôlés par un registre. Chaque échantillonneur est associé à une plage mémoire. Il n'y a pas systématiquement une plage d'adressage particulière qui leur est associée étant donné leurs utilisations diverses. Néanmoins, les échantillonneurs qui capturent des informations de la réception et de la transmission doivent

utiliser les plages de type E pour réaliser l'échantillonnage d'une quantité maximale de données contigües, afin de permettre de caractériser l'amplificateur pour obtenir un modèle de prédistorsion NARMA.

4.5 Conclusion

Ce chapitre a permis de mieux comprendre l'interaction entre les différentes parties du système de prédistorsion conçu et partiellement réalisé. Dans un premier temps, nous avons vu le logiciel de paramétrisation qui s'exécute sur le PC. Les choix faits au niveau de la technologie d'affichage, de l'interaction avec MATLAB et des mécanismes de sauvegarde et de restauration de la configuration ont été vus. Par la suite, le système de contrôle a été expliqué en commençant par les communications TCP/IP entre ce dernier et le logiciel de paramétrisation. L'adressage et le mécanisme de synchronisation ont ensuite été discutés. Finalement, certains des blocs internes les plus importants du système, comme le générateur par table de correspondance, le filtre RRC, le module de prédistorsion et les échantillonneurs ont été vus.

Le système est conçu pour offrir une grande flexibilité à son utilisateur tout en minimisant le temps nécessaire pour configurer le système. Hormis la flexibilité, la performance du système et la facilité de le modifier par d'autres chercheurs ont été des facteurs importants dans les différents choix effectués.

CONCLUSION

Premièrement, nous avons vu des notions théoriques au sujet des caractéristiques importantes des amplificateurs de puissance RF comme la linéarité et l'effet mémoire. Nous avons ensuite vu un aperçu des techniques possibles de reconfiguration hardware. Par la suite, diverses techniques de linéarisation par prédistorsion numérique ont été étudiées. Enfin, l'applicabilité de deux techniques de type NARMA dans un milieu de développement a été comparée.

En second lieu, une architecture d'amplificateur RF reconfigurable a été présentée et sa caractéristique non monotone a été mise en lumière. Nous avons ensuite vu les problèmes que la non monotonie pose vis-à-vis de la prédistorsion d'un tel amplificateur. Une analyse mathématique de la méthode basée sur une structure NARMA a été faite et une nouvelle équation explicite a été présentée. La technique a ensuite été testée en simulation et a confirmé son fonctionnement.

Finalement, un système de prédistorsion conçu et partiellement réalisé a été présenté. Divers choix technologiques de son logiciel de paramétrisation ont été justifiés. Le design des communications, le rôle du système de contrôle, l'adressage et la synchronisation d'horloge réalisée ont été expliqués. Enfin, certains des blocs internes les plus importants ont été détaillés.

RECOMMANDATIONS

- Pour diverses températures ou tensions d'alimentation, stocker le contenu des BPC pour les réutiliser ultérieurement et ainsi accélérer le processus d'adaptation.

- À faible puissance, les effets mémoires sont généralement plus faibles. Lorsque la puissance d'entrée est faible, désactiver un ou des BPC pour réduire la puissance consommée par le système.

ANNEXE I

Extrait de la fiche technique du transistor HBFP-0450

Agilent Technologies
Innovating the HP Way

High Performance Isolated Collector Silicon Bipolar Transistor

Technical Data

HBFP-0450

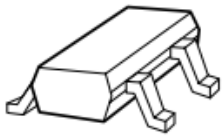
Features

- Ideal for High Performance, Medium Power, and Low Noise Applications
- Typical Performance at 1.8 GHz

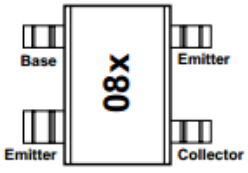
  Medium Power Application
  P_{1dB} of 19 dBm, Noise Figure of 1.7 dB, and Associated Gain of 15 dB at 3 V and 50 mA

  Low Noise Application
  Noise Figure of 1.2 dB, Associated Gain of 13 dB, and P_{1dB} of 11 dBm at 2 V and 10 mA
- Miniature 4-lead SC-70 (SOT-343) Plastic Package
- Transition Frequency f_T = 25 GHz

4-lead SC-70 (SOT-343) Surface Mount Plastic Package



Pin Configuration



Note:
Package marking provides orientation and identification.
08 = Device code
x = Date code character. A new character is assigned for each month, year

Description

Agilent's HBFP-0450 is a high performance isolated collector silicon bipolar junction transistor housed in a 4-lead SC-70 (SOT-343) surface mount plastic package.

This product is based on a 25 GHz transition frequency fabrication process, which enables the products to be used for high performance, medium power, low noise applications up to 6 GHz.

Applications

- Driver amplifier for *Cellular and PCS base stations*
- Driver amplifier and medium power amplifier for *Cellular and PCS handsets*
- High dynamic range LNA for *ISM, wireless data, and WLL applications*
- Oscillator, mixer, and LO Buffer applications

HBFP-0450 Absolute Maximum Ratings

Symbol	Parameter	Units	Absolute Maximum[1]
V_{EBO}	Emitter-Base Voltage	V	1.5
V_{CBO}	Collector-Base Voltage	V	15.0
V_{CEO}	Collector-Emitter Voltage	V	4.5
I_C	Collector Current	mA	100
P_T	Power Dissipation[2]	mW	450
T_j	Junction Temperature	°C	150
T_{STG}	Storage Temperature	°C	-65 to 150

Thermal Resistance:
$\theta_{jc} = 180°C/W$

Notes:
1. Operation of this device above any one of these parameters may cause permanent damage.
2. P_T due to Maximum Ratings.
3. Thermal resistance measured using Liquid Crystal Measurement method.

Electrical Specifications, $T_C = 25°C$

Symbol	Parameters and Test Conditions		Units	Min.	Typ.	Max.
DC Characteristics						
BV_{CEO}	Collector-Emitter Breakdown Voltage	I_C = 1 mA, open base	V	4.5		
I_{CBO}	Collector-Cutoff Current	V_{CB} = 5 V, I_E = 0	nA			500
I_{EBO}	Emitter-Base Cutoff Current	V_{EB} = 1.5 V, I_C = 0	µA			100
h_{FE}	DC Current Gain	V_{CE} = 2 V, I_C = 20 mA	—	50	80	150
RF Characteristics						
P_{-1dB}	Power Output at 1 dB Compression Point	I_C = 50 mA, V_{CE} = 3 V, f = 1.8 GHz I_C = 50 mA, V_{CE} = 2 V, f = 1.8 GHz	dB		19 17	
IP_3	3rd Order Intercept Pt at Output	I_C = 50 mA, V_{CE} = 3 V, f = 1.8 GHz	dBm		29	
G_{-1dB}	Gain at 1 dB Compression Point	I_C = 50 mA, V_{CE} = 3 V, f = 1.8 GHz I_C = 50 mA, V_{CE} = 2 V, f = 1.8 GHz	dBm		16 15.5	
F_{MIN}	Minimum Noise Figure	I_C = 50 mA, V_{CE} = 3 V, f = 1.8 GHz I_C = 50 mA, V_{CE} = 2 V, f = 1.8 GHz	dB		1.7 1.8	
G_a	Associated Gain	I_C = 50 mA, V_{CE} = 3 V, f = 1.8 GHz I_C = 50 mA, V_{CE} = 2 V, f = 1.8 GHz	dB		15 14.5	
NF	Minimum Noise Figure	I_C = 10 mA, V_{CE} = 2 V, f = 1.8 GHz I_C = 20 mA, V_{CE} = 2 V, f = 1.8 GHz	dB		1.2 1.3	 1.7
G_a	Associated Gain	I_C = 10 mA, V_{CE} = 2 V, f = 1.8 GHz I_C = 20 mA, V_{CE} = 2 V, f = 1.8 GHz	dB	 13.0	13 14	
P_{-1dB}	Power Output at 1 dB Compression Point	I_C = 10 mA, V_{CE} = 2 V, f = 1.8 GHz I_C = 20 mA, V_{CE} = 2 V, f = 1.8 GHz	dBm		11 14	

HBFP-0450 Typical Performance, continued

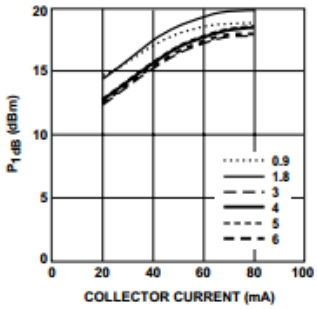


Figure 7. P_{1dB} vs. Collector Current and Frequency.

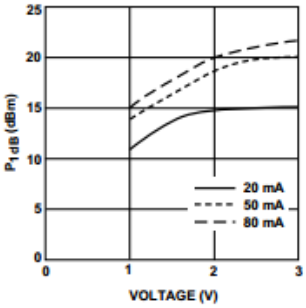


Figure 8. P_{1dB} vs. Voltage at 1.8 GHz.

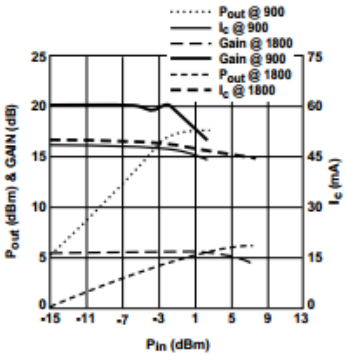


Figure 9. P_{out} (dBm), Gain (dB), and I_c (mA) vs. P_{in} (dBm) at 2 V, 50 mA.
900 MHz: Γ_S: Mag: 0.68, Ang: 121°; Γ_L: Mag: 0.38, Ang: 171°
1800 MHz: Γ_S: Mag: 0.44, Ang: 158°; Γ_L: Mag: 0.28, Ang: 159°

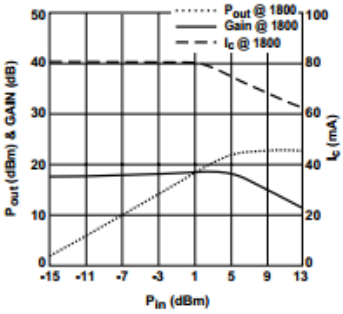


Figure 10. P_{out} (dBm), Gain (dB), and I_c (mA) vs. P_{in} (dBm) at 3 V, 80 mA.
Γ_S: Mag: 0.72, Ang: 169°
Γ_L: Mag: 0.26, Ang: 168°

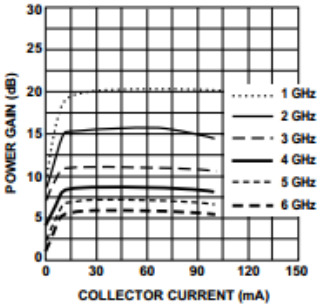


Figure 11. Power Gain vs. Collector Current and Frequency at 2V.

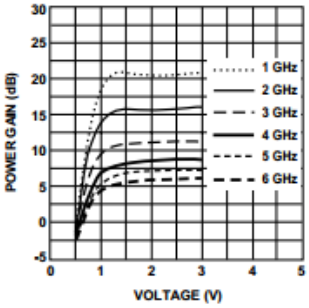


Figure 12. Power Gain vs. Voltage and Frequency at 50 mA.

ANNEXE II

Article RWS 2013

Référence : Thibodeau, A., A. Kouki et N. G. Constantin. 2013. « NARMA-based linearization of RF power amplifiers with non-monotonic response under dynamic hardware reconfiguration ». In 2013 IEEE Radio and Wireless Symposium (RWS). (Austin, TX, 20-23 Jan. 2013), p. 85-87. http://dx.doi.org/10.1109/RWS.2013.6486649

NARMA-Based Linearization of RF Power Amplifiers with Non-Monotonic Response under Dynamic Hardware Reconfiguration

A. Thibodeau, A. Kouki, and N.G. Constantin

École de technologie supérieure, Montréal, Québec, H3C 1K3, Canada
(alexandre.thibodeau.1@ens.etsmtl.ca)

***Abstract* — This paper presents a signal processing block intended for a NARMA-based linearization of an RF power amplifier that exhibits a non-monotonic voltage response. This type of response may be obtained when applying a dynamic hardware reconfiguration of the amplifier for power efficiency improvement, through the activation or deactivation of sections in the RF transistor arrays or the electronic tuning of the bias circuits and the impedance matching circuits, as a function of the instantaneous envelope power of the modulated RF signal. A reconfigurable power amplifier design is described as an example of conditions that introduce a non-monotonic voltage response. The need, in that case, to introduce a new function inversion process for NARMA-based predistortion, is highlighted and explained. The proposed inversion process is described mathematically and validated through a power amplifier linearization example.**

***Index Terms* — Power amplifier, NARMA, linearization, inverse, dynamic reconfiguration.**

I. Introduction

The concept of switching on and off transistor cells within the RF transistor array of an RF power amplifier (PA) depending on the instantaneous amplitude of the envelope of the modulated RF signal being amplified, for the purpose of improving the power added efficiency (PAE) of the PA, has been suggested in the literature. In [1] (Fig. 6), it is suggested that a certain number of cells grouped by appropriate sizes may be switched on and off through digital to analog interface circuitry between an amplitude control word and the RF transistor array, as part of a digital amplitude control and linearization scheme that includes a separate phase modulation of the RF signal.

Part of our work focuses on a similar but simpler technique intended for the PAE improvement of stand-alone RFIC PA modules, requiring analog-only control and switching circuitry simple enough to suit single-chip integration, and used in conjunction with digital signal processing (DSP) outside the PA module for linearity improvement. To reduce the digital signal processing complexity and the associated current consumption, our approach relies on the *soft-switching* of the RF transistor cells and the *gradual* tuning of the biasing and impedance matching circuits with the time varying envelope amplitude. It remains that the AM-AM and AM-PM distortion introduced through these hardware reconfiguration mechanisms require DSP based linearization. For this purpose, we use the NARMA-based (Nonlinear Auto Regressive Moving Average) linearization approach described in [2] because with this approach, a PA characterization that is performed separately is sufficient (i.e. it does not require an embedded characterization process during experimental investigations, but only the PA input to output mapping data). Also, it has the convenience of easier interfacing between an experimental FPGA implementation with a circuit simulation environment (e.g. ADS™) since the characterization process presented in [2] inherently provides a model of the PA.

However, the PA architectures under investigation in our work exhibit a non-monotonic input to output voltage response, and the input to output inversion process (a necessary step in the above mentioned NARMA approach) as proposed in [2] does not allow dealing with the case of a non-monotonic response.

In this paper, we first present a PA design example that exhibits a non-monotonic voltage response upon dynamic hardware reconfiguration. We then formulate the analytical description of an inversion process that is necessary for the linearization of the non-monotonic response with the NARMA approach presented in [2], and provide simulation results of a linearization example to validate the applicability of the inversion process.

II. Reconfigurable Amplifier Architecture

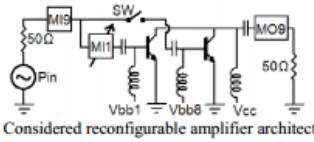


Fig. 1 Considered reconfigurable amplifier architecture.

The architecture considered, illustrated in Fig. 1, consists of two RF transistor cells. The first cell (C1) contains only one silicon bipolar transistor (HBFP-0450 from Avago Technologies), while the second cell (C8) contains 8 in parallel. The global input (MI9) and output (MO9) matching circuits are designed to ensure a maximum output power and a return loss below 25 dB at high power when both C1 and C8 are in operation. These matching circuits remain unchanged at all power levels.

At low power, only C1 is enabled, with a bias current adjusted to 80 mA to meet the gain requirement. The switch (SW) is open and C8 is disabled. Since MI9 is not designed for low power, the matching circuit at the input of C1 (MI1) is electronically adjusted to keep the return loss below 25 dB.

As the input envelope amplitude increases, from around 0.7 V to 1.8 V, the amplifier begins to transition to its high power state. The total bias current of C8 is gradually increased up to 160 mA (i.e. 20 mA per transistor), while that of C1 is decreased to 20 mA. SW is also gradually closed (thanks to the control of the channel resistance of a FET transistor) and MI1 is adjusted at the same time.

At high power, the switch is closed, and both C1 and C8 are enabled. Under these conditions, MI1 is adjusted to have no effect, since MI9 and MO9 are optimally designed for this operating condition.

The response of the circuit can be observed in Fig. 2. It begins on the low power state curve. The reconfiguration starts from 0.7 V input voltage. As it takes place, the output voltage drops until it reaches a minimum. From there, it finally increases until it reaches the high power state. The result is a non-monotonic voltage response, which corresponds to the AM-AM response.

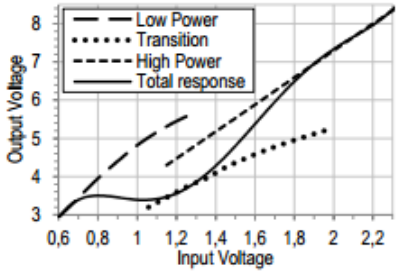


Fig. 2 Voltage input to output response of the proposed reconfigurable amplifier architecture.

III. NARMA-Based Predistortion

The linearization process described in [2] begins by the characterization of the power amplifier with a NARMA-based structure. For that purpose, an LMS (Least Mean Square) filter algorithm is executed on each internal block, using the amplifier input and output voltage data. The direct block F_0 represented in Fig. 3 captures the AM-AM and AM-PM polynomial description of the amplifier. The next step consists in inverting the amplifier model to get the NARMA-based predistorter. The NARMA structure has the convenience of requiring the mathematical inversion of the direct block F_0 only. All the delayed or recursive blocks (not shown in Fig. 3) are computed without inversion. However, the direct block has to be inverted to get F_0^{-1}, which is the function inverse of F_0 (not to be confused with exponentiation).

A. Existing inverse-finding techniques

In [3], the authors present an inversion process which consists in using an LMS filter with inverted input-output with respect to the block F_0, for which we need the inverse function, F_0^{-1}. A polynomial description of $\widehat{G}_0$ is then obtained. Finally, the polynomial is evaluated in many points and converted into a lookup-table (LUT). While this is a very straight-forward approach, it is limited to monotonic responses. In fact, the inverse of a non-monotonic function is not a function. As a consequence, it cannot be represented by a polynomial, which prevents LMS from converging to a correct polynomial description of the inverse.

In [4], the problem of inverting a non-monotonic polynomial is avoided, since it is a look-up table based computation. However, the formulation in [4, eq. (12)] is an implicit relation: for a given input, we don't directly get the corresponding predistortion value(s). Thus, the use of some numerical method is required. No deterministic process is suggested to solve that relation. Hence, over the range of values of the PA input amplitude ($|\hat{z}|$ in Fig. 3), the formulation in [4] does not allow finding the final solution of $\widehat{G}_0$ from a direct and single computation with $|\hat{z}|$ as the only variable of an explicit function. This may have a negative impact on the computation time for a training sequence during a NARMA characterization of the PA in an embedded, real time application.

B. Proposed inverse-finding technique

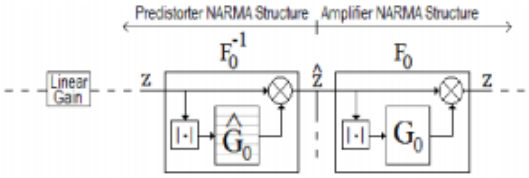


Fig. 3 Relation between the amplifier model direct block F_0 and its corresponding inverse F_0^{-1} in the predistorter.

In order to explain our proposed inversion technique which is based on a deterministic function, we take a closer look at the relation between the direct block and its inverse. From Fig. 3, we can define

$$z = F_0(\hat{z}) = \hat{z}G_0(|\hat{z}|) \quad (1)$$

$$\hat{z} = F_0^{-1}(z) = z\widehat{G}_0(|z|). \quad (2)$$

Substituting (1) in (2), replacing $|\hat{z}|$ by x (scalar), replacing $x|G_0(x)|$ by w(x) (scalar) and rearranging

the terms, we get the following relation:

$$\widehat{G}_0(x|G_0(x)|)=\widehat{G}_0(w(x))=\frac{1}{G_0(x)}. \quad (3)$$

Unlike [4], we get an explicit relation. $G_0(x)$ is the polynomial that was found by LMS during the characterization process. For any value of x that we set (over the supported input voltage range) we can directly calculate the value of $w(x)$ and $\widehat{G}_0(w(x))$. By taking the first derivative of $w(x)$, we can demonstrate that when $|G_0(x)|$ is non-monotonic, $w(x)$ is also non-monotonic, which means that $w(x)$ does not always increase as x increases from the lowest value. In that case, $\widehat{G}_0$ is not a function but a parametric curve: there will be cases where $\widehat{G}_0$ has multiple values for one value of $w(x)$. Consider the case where $w(x_1)=w(x_2)$, we have

$$x_1|G_0(x_1)|=x_2|G_0(x_2)|. \quad (4)$$

From (3) and (4), where $x_1<x_2$, we get

$$|G_0(x_1)|>|G_0(x_2)| \Rightarrow |\widehat{G}_0(w(x_1))|<|\widehat{G}_0(w(x_2))|. \quad (5)$$

As a result, while x increases, $w(x)$ can decrease, but the first pass over an abscissa always yields the lowest value for $\widehat{G}_0(w(x))$. In other words, the first predistortion value found is always the lowest for a given expected output value. By varying x over the range of input values from the minimum to the maximum, we directly calculate the inverse function in a parametric fashion. With this systematic approach, for each new calculated value of $w(x)$ and $\widehat{G}_0(w(x))$, we choose the ones corresponding to the lowest x, hence the lowest value of $\widehat{G}_0(w(x))$, (implying the lowest input amplitude to the PA to minimize distortion), and discard the higher values. Finally, the results can be stored in the LUT of $\widehat{G}_0$.

IV. Experimental Validation

To validate the proposed technique, we used the reconfigurable amplifier described earlier. Taking its non-monotonic voltage response as a black box, we used LMS to find the 8th order polynomial model of the direct block in the NARMA-based structure. The input voltage ranged from 0.6 V to 2.3 V. Once the model identified by LMS, we executed the algorithm described in section III.B.

With the LUT of the predistorter block now populated, we used the predistorter to linearize the constellation displayed in Fig. 4. The resulting constellation shows a great improvement over the non-linearized one. In fact, the points of the linearized constellation practically superimpose with those of the expected constellation, showing that the linearization process was performed successfully.

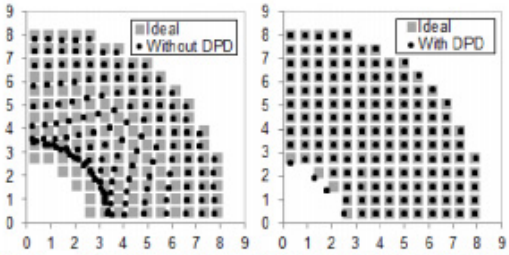


Fig. 4 Comparison of the resulting constellation without/with digital predistortion (DPD) and the expected constellation.

V. Conclusion

In this paper, we presented a reconfigurable power amplifier architecture that exhibits a non-monotonic voltage response. In the context of NARMA-based predistortion, we explained how previously published implementations of the required function inversion cannot deal with non-monotonic responses or require the use of numerical methods. We presented a method that allows finding directly the inverse of a non-monotonic polynomial of the NARMA-based amplifier model. The algorithm has been successfully verified on a reconfigurable amplifier architecture that exhibits such a behavior.

Acknowledgement

The authors wish to acknowledge the support of NSERC, CMC Microsystems and to thank David Berthiaume for help with simulation.

References

[1] P. Asbeck, L. Larson, D. Kimball, S. Pornpromlikit, J.-H. Jeong, C. Presti, T. P. Hung, F. Wang, and Y. Zhao, "Design options for high efficiency linear handset power amplifiers," *IEEE Topical Meeting on Silicon Monolithic Integrated Circuits in RF Systems*, pp. 1-4, Jan. 2009.

[2] P. L. Gilabert, A. Cesari, G. Montoro, E. Bertran, and J. M. Dilhac, "Multi-Lookup Table FPGA Implementation of an Adaptive Digital Predistorter for Linearizing RF Power Amplifiers With Memory Effects," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, pp. 372-384, 2008.

[3] G. Montoro, P. L. Gilabert, E. Bertran, A. Cesari, and D. D. Silveira, "A New Digital Predictive Predistorter for Behavioral Power Amplifier Linearization," *IEEE Microwave and Wireless Components Letters*, vol. 17, pp. 448-450, 2007.

[4] L. Ning, K. Xiaowei, X. Wei, and H. Zishu, "An LMS-based close-loop digital predistorter for RF power amplifiers using NARMA structure," *2010 2nd International Conference on Signal Processing Systems (ICSPS)*, 2010, pp. V3-605-V3-608.

ANNEXE III

Implémentation de l'algorithme LMS pour MATLAB

```
function theta = LMS(data, cfg, theta)
    % Short-parameters
    P = cfg.P;
    nCoeff = cfg.nCoeff;
    nFIR = cfg.nFIR;
    nRIR = cfg.nRIR;
    spp = cfg.spp;
    mu = cfg.mu;

    input = data(:,1);  % Complex I/Q before modulation and PA.
    output = data(:,2); % Complex I/Q after PA and demodulation.
    nPackets = floor(size(data,1)/spp); % Number of packets.

    % For each packet...
    for iPacket = 1:nPackets
        % Get the data.
        x = input(1+(iPacket-1)*spp:iPacket*spp);
        y = output(1+(iPacket-1)*spp:iPacket*spp);

        % Calculate the absolute values.
        xabs = abs(x);
        yabs = abs(y);

        % Calculate the intermediate powers.
        xp = zeros(spp,nCoeff);
        yp = zeros(spp,nCoeff);
        for p = 0:P
            xp(:,p+1) = x.*xabs.^p;
            yp(:,p+1) = y.*yabs.^p;
        end

        % Initialize theta.
        %theta = zeros(nCoeff*(nFIR+nRIR+1),1);

        % For each time step...
        % We might need to use previous samples.
        kmin = max(nFIR,nRIR) + 1;

        for k = kmin:spp
            % Initialize u.
            u = zeros(nCoeff*(nFIR+nRIR+1),1);

            % Copy xp.
            for i = 0:nFIR
                u(1+i*nCoeff:(i+1)*nCoeff) = xp(k-i,:);
            end
```

```
            % Copy yp.
            offset = (nFIR+1)*nCoeff;
            for i = 1:nRIR
                u(1+offset+(i-1)*nCoeff:offset+i*nCoeff) = yp(k-i,:);
            end

            % Define yc.
            yc = theta'*u;

            % Define err.
            err = y(k)-yc;

            % Adjust theta.
            if(mu>2/sum(abs(u).^2))
              %warning('Mu is too high!.')
              mu = 0.99*2/sum(abs(u).^2)
            end
            theta = theta + mu*u*conj(err);
        end
    end
end
```

ANNEXE IV

Implémentation de la technique d'inversion par équation explicite pour MATLAB

```
% Parameters
dw_min = 0.0006;
dw_max = 0.0012;
dx = 0.001;
maxsteps = 10000;
w_max = 1.01;

% Invert
G = [0 0i];
x_prev = 0;
w_prev = 0;
w_lastsave = 0;
j = 0;
for i=1:maxsteps
    if(i > 0.9*maxsteps)
        kk = 1;
    end

    x = x_prev + dx;
    fx = ComputeBPC(x, theta(1:cfg.nCoeff))/x;
    %Note: we divide by x because fx is the LUT,
    %not the whole BPC.

    w=x*abs(fx);
    dw=w-w_prev;
    if abs(dw) > dw_max
        % Too large step in w: reduce the step in x.
        dx = 0.8*dx;
        continue;
    end


    if abs(dw) < dw_min
        % Too small step in w.
        x_prev = x;
        continue;
    end;
```

```
    % Step size in w is good.
    x_prev = x;
    w_prev = w;
    dx = 1.2*dx;
    if (w-w_lastsave > 0)
        % Step is forward: save and increase step.
        w_lastsave = w;
        fw=1/fx;
        G = [G; w, fw];
    end

    if(w > w_max)
        break;
    end
end

figure(3); plot(G(:,1),abs(G(:,2)))
figure(4); plot(G(:,1),angle(G(:,2)))
```

ANNEXE V

Utilisation de liaisons de données dans le contrôle RRCFilterSettings

```
<local:GenericSettings
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:PlatformManager.UI.Configuration.Blocks"
    xmlns:PM_Configuration="clr-namespace:PlatformManager.UI.Configuration"
    x:Class="PlatformManager.UI.Configuration.Blocks.RRCFilterSettings"
    mc:Ignorable="d" Height="400" Width="400">

    <TabControl Width="Auto" Height="Auto">
        <TabItem Header="General">
            <Grid>
                <TextBlock HorizontalAlignment="Left"
                    VerticalAlignment="Top"
                    Margin="10,10,0,0" Text="Block Type:" />
                <TextBlock HorizontalAlignment="Stretch"
                    VerticalAlignment="Top"
                    Margin="75,10,10,0" Text="{Binding Type}" />
                <TextBlock HorizontalAlignment="Left"
                    VerticalAlignment="Top"
                    Margin="10,31,0,0" Text="Block Name:" />
                <TextBlock HorizontalAlignment="Stretch"
                    VerticalAlignment="Top"
                    Margin="81,31,10,0" Text="{Binding Name}" />
                <TextBlock HorizontalAlignment="Left"
                    VerticalAlignment="Top"
                    Margin="10,52,0,0" Text="Comments:" />
                <TextBox HorizontalAlignment="Stretch"
                    VerticalAlignment="Stretch"
                    Margin="10,73,10,10" Text="{Binding Path=Comments}" />
            </Grid>
        </TabItem>
[...]
```

ANNEXE VI

Utilisation de commandes dans la fenêtre GenericSettingsWindow

```
<Window x:Class="PlatformManager.UI.Configuration.Blocks.GenericSettingsWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:PlatformManager.UI.Configuration.Blocks"
    Title="Configuration - ???" Height="500" Width="500">

    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.Save"
                        Executed="SaveCommand" />
        <CommandBinding Command="ApplicationCommands.Close"
                        Executed="CloseCommand" />
    </Window.CommandBindings>

    <DockPanel LastChildFill="True">
        <Grid DockPanel.Dock="Bottom" Height="42">
            <Button Content="OK"
                    HorizontalAlignment="Right"
                    Margin="0,10,90,0"
                    VerticalAlignment="Top"
                    Width="75"
                    Command="ApplicationCommands.Save"/>
            <Button Content="Cancel"
                    HorizontalAlignment="Right"
                    Margin="0,10,10,0"
                    VerticalAlignment="Top"
                    Width="75"
                    Command="ApplicationCommands.Close"/>
        </Grid>
        <DockPanel Name="DynamicContainer" LastChildFill="True" />
    </DockPanel>
</Window>
```

ANNEXE VII

Interface de sérialisation/désérialisation XML

```
using System.Xml;

namespace PlatformManager.Tools
{
    public interface IFlexibleXmlSerializable
    {
        /// <summary>
        /// Deserializes an XML Element into this object.
        /// </summary>
        /// <param name="iCurrent">XML Element that serializes
        /// this object.</param>
        void Deserialize(XmlElement iCurrent);

        /// <summary>
        /// Serializes this object into a parent XML Element.
        /// </summary>
        /// <param name="iParent">XML Parent.</param>
        /// <returns>XML Element that serializes this object.
        /// Typically used for inheritance.</returns>
        void Serialize(XmlElement iCurrent);
    }
}
```

ANNEXE VIII

Classe de sérialisation XmlSerializationHelper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;

namespace PlatformManager.Tools
{
  public class XmlSerializationHelper
  {
    protected static XmlSerializationHelper mInstance;

    /// <summary>
    /// Singleton implementation.
    /// </summary>
    public static XmlSerializationHelper Instance
    {
      get
      {
        if (mInstance == null)
        {
          mInstance = new XmlSerializationHelper();
        }
        return mInstance;
      }
    }

    /// <summary>
    /// Creates a child node with the specified name to the specified XML parent node.
    /// </summary>
    public XmlElement CreateElement(XmlNode iParent, string iName)
    {
      return CreateElement(iParent, iName, null);
    }

    /// <summary>
    /// Creates an XML Element that will be the child of iParent.
    /// </summary>
    /// <param name="iParent">Parent of the element created.</param>
    /// <param name="iName">Name of the element.</param>
    /// <param name="iType">When non-null, will be used to assign the xsi:Type
    /// attribute.</param>
    /// <returns>New Child.</returns>
    public XmlElement CreateElement(XmlNode iParent, string iName, Type iType)
    {
      XmlElement wElement;
      if (iParent is XmlDocument)
      {
        wElement = ((XmlDocument)iParent).CreateElement(iName);
      }
      else
      {
        wElement = iParent.OwnerDocument.CreateElement(iName);
      }

      if (iType != null)
      {
        XmlAttribute wTypeAttribute =
          wElement.OwnerDocument.CreateAttribute("serializationType");
```

```
    wTypeAttribute.Value = iType.FullName;
    wElement.Attributes.Append(wTypeAttribute);
  }

  iParent.AppendChild(wElement);

  return wElement;
}

/// <summary>
/// Adds a string attribute to a XML element.
/// </summary>
/// <param name="iElement">XML element that will get the new attribute.</param>
/// <param name="iName">Name of the attribute.</param>
/// <param name="iValue">String value of the attribute.</param>
/// <returns></returns>
public XmlAttribute CreateAttributeString(XmlNode iElement, string iName, string iValue)
{
  XmlAttribute wAttribute = iElement.OwnerDocument.CreateAttribute(iName);
  wAttribute.Value = iValue;
  iElement.Attributes.Append(wAttribute);
  return wAttribute;
}

/// <summary>
/// Adds an attribute to a XML element from a simple value
/// that can be converted to a string.
/// </summary>
/// <typeparam name="T">Type of the value.</typeparam>
/// <param name="iElement">XML element that will get the new attribute.</param>
/// <param name="iName">Name of the attribute.</param>
/// <param name="iValue">Value of the attribute.</param>
/// <returns></returns>
public XmlAttribute CreateAttributeOfConvertible<T>(XmlNode iElement,
                                                    string iName, T iValue)
{
  System.ComponentModel.TypeConverter wConverter =
    System.ComponentModel.TypeDescriptor.GetConverter(typeof(T));
  if (wConverter != null)
  {
    XmlAttribute wAttribute = iElement.OwnerDocument.CreateAttribute(iName);
    wAttribute.Value = wConverter.ConvertToString(iValue);
    iElement.Attributes.Append(wAttribute);
    return wAttribute;
  }
  return null;
}

/// <summary>
/// Serializes a collection of complex elements.
/// </summary>
/// <typeparam name="T">Common class of the items in the collection.</typeparam>
/// <param name="iList">XML element of the list.</param>
/// <param name="iCollection">Collection of items.</param>
public void SerializeCollectionOfSerializable<T>(XmlNode iList,
                                                  ICollection<T> iCollection)
  where T : class, IFlexibleXmlSerializable
{
  foreach (IFlexibleXmlSerializable wItem in iCollection)
  {
    XmlElement wElement = CreateElement(iList, "Item", wItem.GetType());
    wItem.Serialize(wElement);
  }
}
```

```
    /// <summary>
    /// Serializes a collection of simple elements.
    /// </summary>
    /// <typeparam name="T">Common class of the items in the collection.</typeparam>
    /// <param name="iList">XML element of the list.</param>
    /// <param name="iCollection">Collection of items.</param>
    public void SerializeCollectionOfConvertible<T>(XmlNode iList,
                                                    ICollection<T> iCollection)
    {
      System.ComponentModel.TypeConverter wConverter =
        System.ComponentModel.TypeDescriptor.GetConverter(typeof(T));
      if (wConverter != null)
      {
        foreach (T wItem in iCollection)
        {
          XmlElement wElement = CreateElement(iList, "Item");
          wElement.InnerText = wConverter.ConvertToString(wItem);
        }
      }
    }
  }
}
```

ANNEXE IX

Classe de désérialisation XmlDeserializationHelper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Xml;

namespace PlatformManager.Tools
{
    public class XmlDeserializationHelper
    {
        protected static XmlDeserializationHelper mInstance;

        /// <summary>
        /// Singleton implementation.
        /// </summary>
        public static XmlDeserializationHelper Instance
        {
            get
            {
                if (mInstance == null)
                {
                    mInstance = new XmlDeserializationHelper();
                }
                return mInstance;
            }
        }

        /// <summary>
        /// For the specified XML element, fetches the content of the attribute with
        /// the specified name. If the attribute does not exist, the default is returned.
        /// </summary>
        public string FetchAttributeString(XmlElement iElement,
                                           string iName,
                                           string iDefault)
        {
            if (iElement != null)
            {
                XmlAttribute wAttribute = iElement.SelectSingleNode("./@" + iName)
                    as XmlAttribute;
                if (wAttribute != null)
                {
                    return wAttribute.Value;
                }
            }
            return iDefault;
        }
```

```
/// <summary>
/// Convert an attribute to its original value from a string stored in an attribute.
/// If the attribute cannot be found, the default value is returned.
/// </summary>
/// <typeparam name="T">Type of the value.</typeparam>
/// <param name="iElement">Element that has the attribute.</param>
/// <param name="iName">Name of the attribute.</param>
/// <param name="iDefault">Default value.</param>
/// <returns></returns>
public T FetchAttributeOfConvertible<T>(XmlElement iElement,
                                        string iName,
                                        T iDefault)
{
    if (iElement != null)
    {
        System.ComponentModel.TypeConverter wConverter =
            System.ComponentModel.TypeDescriptor.GetConverter(typeof(T));
        if (wConverter != null)
        {
            XmlAttribute wAttribute = iElement.SelectSingleNode("./@" + iName)
                as XmlAttribute;
            if (wAttribute != null)
            {
                return (T)wConverter.ConvertFromString(wAttribute.Value);
            }
        }
    }
    return iDefault;
}

/// <summary>
/// Fetches an XML element from its parent by its name.
/// </summary>
/// <param name="iParent">Parent that should contain the element.</param>
/// <param name="iName">Name of the element to be found.</param>
/// <returns></returns>
public XmlElement FetchElement(XmlElement iParent, string iName)
{
    return FetchElement(iParent, iName, null);
}

/// <summary>
/// Fetches an XML element from its parent by its name and deserializes it
/// to a specified target.
/// </summary>
/// <param name="iParent">Parent that should contain the element.</param>
/// <param name="iName">Name of the element to be found.</param>
/// <param name="iObject">Deserialization target.</param>
/// <returns></returns>
public XmlElement FetchElement(XmlElement iParent, string iName,
    IFlexibleXmlSerializable iObject)
{
    XmlElement wElement = iParent.SelectSingleNode("./" + iName) as XmlElement;
    if (wElement != null && iObject != null)
    {
        iObject.Deserialize(wElement);
    }
    return wElement;
}
```

```
/// <summary>
/// Fetches the string content of a specified XML element from its parent node.
/// </summary>
public string FetchElementString(XmlElement iParent, string iName, string iDefault)
{
    XmlElement wElement = iParent.SelectSingleNode("./" + iName) as XmlElement;
    if (wElement != null)
    {
        return wElement.Value;
    }
    return iDefault;
}

/// <summary>
/// Deserializes a collection of complex objects.
/// </summary>
/// <typeparam name="T">Storage class of the collection.</typeparam>
/// <param name="iList">XML element of the list.</param>
/// <param name="iCollection">Deserialization target.</param>
public void DeserializeCollectionOfSerializable<T>(XmlNode iList,
                                                   ICollection<T> iCollection)
    where T : class, IFlexibleXmlSerializable
{
    iCollection.Clear();
    if (iList != null)
    {
        foreach (XmlElement wItem in
                 iList.SelectNodes("./Item").OfType<XmlElement>())
        {
            string wTypeName = FetchAttributeString(wItem, "serializationType", "");
            if (wTypeName != "")
            {

                Type wType = Type.GetType(wTypeName);
                if (wType != null)
                {
                    T wObject = Activator.CreateInstance(wType) as T;
                    if (wObject != null)
                    {
                        wObject.Deserialize(wItem);
                        iCollection.Add(wObject);
                    }
                }
            }
        }
    }
}
```

```
        /// <summary>
        /// Deserializes a collection of simple objects.
        /// </summary>
        /// <typeparam name="T">Storage class of the collection.</typeparam>
        /// <param name="iList">XML element of the list.</param>
        /// <param name="iCollection">Deserialization target.</param>
        public void DeserializeCollectionOfConvertible<T>(XmlNode iList,
                                                          ICollection<T> iCollection)
        {
            iCollection.Clear();
            if (iList != null)
            {
                System.ComponentModel.TypeConverter wConverter =
                    System.ComponentModel.TypeDescriptor.GetConverter(typeof(T));
                if (wConverter != null)
                {
                    foreach (XmlElement wItem in
                        iList.SelectNodes("./Item").OfType<XmlElement>())
                    {
                        iCollection.Add((T)wConverter.ConvertFromString(wItem.InnerText));
                    }
                }
            }
        }
    }
}
```

ANNEXE X

Sérialisation/désérialisation de la classe RRCFilter

```
#region IFlexibleXmlSerializable Members

public override void Deserialize(XmlElement iCurrent)
{
    base.Deserialize(iCurrent);
    XmlDeserializationHelper wXDH = XmlDeserializationHelper.Instance;
    ScriptFile = wXDH.FetchAttributeString(iCurrent, "scriptFile", "");
    ScriptArguments = wXDH.FetchAttributeString(iCurrent, "scriptArguments", "");
    wXDH.DeserializeCollectionOfConvertible<double>(wXDH.FetchElement(iCurrent,
        "Coefficients"), Coefficients);
}

public override void Serialize(XmlElement iCurrent)
{
    base.Serialize(iCurrent);
    XmlSerializationHelper wXSH = XmlSerializationHelper.Instance;
    wXSH.CreateAttributeString(iCurrent, "scriptFile", ScriptFile);
    wXSH.CreateAttributeString(iCurrent, "scriptArguments", ScriptArguments);
    wXSH.SerializeCollectionOfConvertible<double>(wXSH.CreateElement(iCurrent,
        "Coefficients"), Coefficients);
}

#endregion
```

ANNEXE XI

Sérialisation/désérialisation de la classe Block2x2

```
#region IFlexibleXmlSerializable Members

public override void Deserialize(XmlElement iCurrent)
{
    base.Deserialize(iCurrent);
    XmlDeserializationHelper wXDH = XmlDeserializationHelper.Instance;
    PortI_In.Deserialize(wXDH.FetchElement(iCurrent, "PortI_In"));
    PortQ_In.Deserialize(wXDH.FetchElement(iCurrent, "PortQ_In"));
    PortI_Out.Deserialize(wXDH.FetchElement(iCurrent, "PortI_Out"));
    PortQ_Out.Deserialize(wXDH.FetchElement(iCurrent, "PortQ_Out"));
}

public override void Serialize(XmlElement iCurrent)
{
    base.Serialize(iCurrent);
    XmlSerializationHelper wXSH = XmlSerializationHelper.Instance;
    PortI_In.Serialize(wXSH.CreateElement(iCurrent, "PortI_In"));
    PortQ_In.Serialize(wXSH.CreateElement(iCurrent, "PortQ_In"));
    PortI_Out.Serialize(wXSH.CreateElement(iCurrent, "PortI_Out"));
    PortQ_Out.Serialize(wXSH.CreateElement(iCurrent, "PortQ_Out"));
}

#endregion
```

ANNEXE XII

Sérialisation/désérialisation de la classe Block

```
#region IFlexibleXmlSerializable Members

public virtual void Deserialize(XmlElement iCurrent)
{
    XmlDeserializationHelper wXDH = XmlDeserializationHelper.Instance;
    Name = wXDH.FetchAttributeString(iCurrent, "name", "<< Name >>");
    Type = wXDH.FetchAttributeString(iCurrent, "type", "<< Type >>");
    Comments = wXDH.FetchAttributeString(iCurrent,
                                         "comments",
                                         String.Empty);
}

public virtual void Serialize(XmlElement iCurrent)
{
    XmlSerializationHelper wXSH = XmlSerializationHelper.Instance;
    wXSH.CreateAttributeString(iCurrent, "name", Name);
    wXSH.CreateAttributeString(iCurrent, "type", Type);
    wXSH.CreateAttributeString(iCurrent, "comments", Comments);
}

#endregion
```

ANNEXE XIII

Extrait d'un fichier XML contenant un résultat de sérialisation

```
<?xml version="1.0" encoding="utf-8"?>
<Platform xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="DemoXml"
        description="This is a demo configuration." matlabPreloadScript="Scripts/Preload.m">
    <RRCFilter1 name="RRCFilter1" type="RRC Filter" comments=""
        scriptFile="Scripts/RRCFilter/RRCFilter.m" scriptArguments="0.5, 128, 8">
    <PortI_In name="I_In" direction="Input" net="" />
    <PortQ_In name="Q_In" direction="Input" net="" />
    <PortI_Out name="I_Out" direction="Output" net="" />
    <PortQ_Out name="Q_Out" direction="Output" net="" />
    <Coefficients>
        <Item>-0,00031206851586646145</Item>
        <Item>-0,00026382210239559416</Item>
        <Item>-0,00011907323538665947</Item>
        <Item>7,9563263831253635E-05</Item>
        <Item>0,00026795128457056746</Item>
        <Item>0,00038111058083413685</Item>
        <Item>0,00037558108408837937</Item>
        <Item>0,000245903896101261</Item>
        <Item>2,9149256976537805E-05</Item>
        <Item>-0,00020490329656540563</Item>
        <Item>-0,0003755810840883803</Item>
        <Item>-0,00041850187493242338</Item>
        <Item>-0,00030917455911988462</Item>
        <Item>-7,4629168201581345E-05</Item>
         [...]
    </Coefficients>
    [...]
</Platform>
```

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Constantin, N. G., K. H. Kwok, Shao Hongxiao, C. Cismaru et P. J. Zampardi. 2012. « Formulations and a Computer-Aided Test Method for the Estimation of IMD Levels in an Envelope Feedback RFIC Power Amplifier ». *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 31, n° 12, p. 1881-1893.

Cripps, Steve C. 2006. *RF power amplifiers for wireless communications*, 2nd. Boston: Artech House.

Francis, Michael. 2009. *Infinite Impulse Response Filter Structures in Xilinx FPGAs*. <http://www.xilinx.com/support/documentation/white_papers/wp330.pdf>. Consulté le 10 juin 2013.

Gilabert, P. L. 2008. « Multi Look-Up Table Digital Predistortion for RF Power Amplifier Linearization ». Ph.D Thesis, Barcelona, Universitat Politècnica de Catalunya, 164 p. <http://hdl.handle.net/10803/6915>.

Gilabert, P. L., A. Cesari, G. Montoro, E. Bertran et J. M. Dilhac. 2008. « Multi-Lookup Table FPGA Implementation of an Adaptive Digital Predistorter for Linearizing RF Power Amplifiers With Memory Effects ». *Microwave Theory and Techniques, IEEE Transactions on,* vol. 56, n° 2, p. 372-384.

Gilabert, P. L., G. Montoro et E. Bertran. 2005. « On the Wiener and Hammerstein models for power amplifier predistortion ». In *Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings* (4-7 Dec. 2005). Vol. 2, p. 4 pp. In *IEEE Xplore*. New-york: Institute of Electrical and Electronics Engineers.

Gilabert, P. L., G. Montoro et E. Bertran. 2011. « FPGA Implementation of a Real-Time NARMA-Based Digital Adaptive Predistorter ». *Circuits and Systems II: Express Briefs, IEEE Transactions on,* vol. 58, n° 7, p. 402-406.

Gilabert, P. L., G. Montoro et A. Cesari. 2006. « A recursive digital predistorter for linearizing RF power amplifiers with memory effects ». In *Microwave Conference, 2006. APMC 2006. Asia-Pacific* (12-15 Dec. 2006). p. 1040-1043. In *IEEE Xplore*. New-york: Institute of Electrical and Electronics Engineers.

Ginosar, Ran. 2011. « Metastability and Synchronizers: A Tutorial ». *IEEE Design & Test of Computers*, p. 23-35.

IEEE. 1994. « IEEE Standard VHDL Language Reference Manual ». *ANSI/IEEE Std 1076-1993*.

IEEE. 2009. « IEEE Standard VHDL Language Reference Manual ». *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, p. c1-626.

Montoro, G., P. L. Gilabert, E. Bertran, A. Cesari et D. D. Silveira. 2007. « A New Digital Predictive Predistorter for Behavioral Power Amplifier Linearization ». *Microwave and Wireless Components Letters, IEEE,* vol. 17, n^{o} 6, p. 448-450.

Ning, Liu, Kong Xiaowei, Xia Wei et He Zishu. 2010. « An LMS-based close-loop digital predistorter for RF power amplifiers using NARMA structure ». In *Signal Processing Systems (ICSPS), 2010 2nd International Conference on* (5-7 July 2010). Vol. 3, p. V3-605-V3-608.

SungWon, Chung, J. W. Holloway et J. L. Dawson. 2008. « Energy-Efficient Digital Predistortion With Lookup Table Training Using Analog Cartesian Feedback ». *Microwave Theory and Techniques, IEEE Transactions on,* vol. 56, n^{o} 10, p. 2248-2258.

Thibodeau, A., A. Kouki et N. G. Constantin. 2013. « NARMA-based linearization of RF power amplifiers with non-monotonic response under dynamic hardware reconfiguration ». In *Radio and Wireless Symposium (RWS), 2013 IEEE* (20-23 Jan. 2013). p. 85-87. In *IEEE Xplore*. New-York: Institute of Electrical and Electronics Engineers.

Vuolevi, J.; Rahkonen, T. 2003. *Distortion in RF Power Amplifiers*. Artech House. Consulté le 23 mars 2012.