Table des matières

Liste des tableaux

Liste des figures

Liste des symboles

$[]^*$	Conjugué d'un signal complexe
$[]^\dagger$	transposée conjuguée
$[]^\top$	Transposée
$\mathbf{a}$	Vecteur directionnel du réseau
$\mathbf{A}$	Matrice des vecteurs directionnels
d	Distance séparant les capteurs
$\mathbf{D}$	Matrice diagonale contenant les valeurs propres
f_0	fréquence porteuse
$\mathbf{G}$	Matrice de calibration
$\mathbf{I}$	Matrice identité
$\mathbf{J}$	Matrice anti diagonale
k	Indice du temps d'échantillonnage ou de l'échantillon
K	Nombre total d'échantillons
λ_i	Valeurs propres de $\mathbf{R}_{xx}$
m	indice des sources
M	Nombre de sources
N	Indice d'élément du réseau
N	Nombre d'éléments du réseau
$\mathbf{n}$	Vecteur de bruit des signaux
$\mathbf{N}$	Matrice des vecteurs de bruit des signaux
$\mathbf{P}_n$	Projecteur dans le sous-espace bruit
ϕ	Déphasage d'un élément du réseau causé par la différence de parcourt des ondes pour se rendre à un capteur
$\mathbf{Q}$	Simple rotation dans la décomposition EVD
$\mathbf{R}_{xx}$	Matrice de covariance de taille $N \times N$
$\mathbf{R}_{ss}$	Matrice de covariance source $M \times M$
$\mathbf{S}$	Matrice de rotation proposée par Wallace Givens
θ	Amgle d'arrivée du signal incident au réseau
$\mathbf{V}$	Matrice des vecteurs propres
$\mathbf{V}_n$	Matrice des vecteurs propres bruit
$\mathbf{V}_s$	Matrice des vecteurs propres source

U Matrice unitaire

x Vecteur des signaux reçus

X Matrice des signaux reçus

Liste des sigles et acronymes

ADC	Analog-to-Digital Converter
AIC	Akaike Information Criteria
BRAM	Block Random Access Memory
BSP	Board Support Package
BSDK	Board Software Developement Kit
CORDIC	COordinate Rotational DIgital Computer
DOA	Direction Of Arrival
DP	Dual Port
EVD	Eigen Value Decomposition
FIFO	Fisrt Input Fisrt Output
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
LO	Local Oscillator
LRTS	Laboratoire de Radiocommunication et de Traitement du Signal
LUT	Look Up Table
MAPE	Mean Absolute Percentage Error
MBD	Model based Design
MBDK	Model-Based Design Kit
MUSIC	MUltiple SIgnal Classification
PD	Processeur Non Diagonal
PND	Processeur Non Diagonal
PV	Processeur Vectoriel
RAM	Random Access Memory
RMSE	Root Mean Sqaure Error
RF	Radio Fréquence
ULA	Uniform Linear Array
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration
XSG	Xilinx System Generator

Remerciements

Je tiens tout d'abord à remercier mon directeur de recherche Dominic Grenier d'avoir fait le pari d'accepter de diriger mon projet de maitrise et faciliter mon intégration au sein du LRTS. C'est dans l'humilité, la patience, la bonne humeur et la compétence, qu'il a su me faire bénéficier d'un encadrement de qualité remarquable avec une grande marge d'autonomie. Je le remercie également pour l'octroi d'une bourse tirée de ses propres fonds de subvention.

Je souhaiterais remercier *Nutaq*^{TM}, le partenaire à ce projet de maitrise d'avoir fourni le matériel et l'aide technique nécessaire à la complétion de ce projet. En particulier, j'adresse ma gratitude à l'endroit de Tristan Martin gestionnaire au développement des affaires, Jean-Benoit Larouche ingénieur R&D et Jose Quintero. Je remercie aussi toute l'équipe d'ingénierie (support) qui m'a assisté à fixer les problèmes techniques survenus çà et là avec le *PicoDigitizer*. Par la même occasion, je profite pour dire merci aux techniciens du département de génie électrique et informatique de l'université Laval pour leur aide dans l'installation des logiciels utilisés dans le projet.

Remerciement spécial à ma mère et mon père pour leurs prières et leurs bénédictions. Je suis chanceux de les avoir comme parents. À mes frères et sœurs, je dis merci pour tous les mots d'encouragements. Un grand merci à Marie-Rose Luong pour son soutien inestimable dans ma vie quotidienne. Une vraie source de motivation ! À tous, je leur dis merci pour leur amour et leur foi en moi. Je vous aime tous !

Avant de clore ce chapitre de remerciements, je voudrais remercier tous les membres du LRTS pour leur accueil chaleureux et leur convivialité à mon égard. Je n'oublierai jamais tous ces agréables moments partagés en compagnie des professeurs Paul Fortier, Jean-Yves Chouinard, Benoit Gosselin, Amine Miled et mes collègues Mathieu Gallichand, Simon East-Lavoie, Carl Poirier et autres. Cette maitrise a été pour moi à la fois une véritable aventure humaine et professionnelle.

Enfin, en tant que croyant, je remercie Dieu Tout Puissant d'avoir permis que cette maitrise soit une réalité et placé sur mon chemin toutes ces bonnes personnes.

Ce fut un réel plaisir !

Chapitre 1

Introduction

Depuis longtemps, déjà, dans la sphère des applications scientifiques et techniques des antennes, la combinaison des signaux reçus par un réseau d'antennes a été au centre des études d'innombrables ingénieurs et chercheurs. Entre autres, la localisation de l'origine des signaux reçus par un réseau d'antennes. Le principe est de traiter sur la base d'algorithme de localisation, les ondes électromagnétiques propagées dans l'air du point d'émission au réseau d'antennes, en vue d'évaluer leur Direction d'Arrivée (DOA) par rapport au réseau. La DOA demeure le socle d'énumérables applications pratiques dans divers domaines.

Dans le département de la défense, la DOA peut être appliquée pour identifier la direction de menace potentielle. En télécommunications, elle reste incontournable pour certaines techniques permettant d'augmenter les capacités des systèmes de communications face au nombre sans cesse grandissant des utilisateurs. La DOA permet aux stations de base de localiser les usagers, afin de permettre une réutilisation des mêmes fréquences de communications au sein d'une même cellule par la formation des faisceaux distincts pour chacun de ces usagers, sans interférences ou parasites pouvant détériorer la qualité de service de ces systèmes. Cette méthode est appelée Spatial Division Multiple Access ou Accès Multiple à Répartition Spatiale. De plus, la connaissance précise des directions et des délais d'arrivée serait suffisante pour bien estimer le canal de propagation sans avoir recours à un préambule comme cela se fait en communication radiomobile.Un autre exemple d'application est l'identification très précise de la direction d'un appel téléphonique à l'issue d'une urgence de sorte que l'équipe de sauvetage puisse être envoyée à l'endroit approprié. Ce dernier exemple est d'autant plus pratique, car contrairement aux technologies actuellement utilisées telles que le GPS et la triangulation des stations de bases, l'on serait en mesure d'identifier clairement un appartement dans un bloc d'immeuble même à plusieurs étages.

Plusieurs algorithmes de détermination de la direction d'arrivée de faisceaux par rapport au réseau d'antennes existent déjà et chacun a son avantage et inconvénient. Le principe général de fonctionnement de ces algorithmes est d'obtenir, à partir des données recueillies, un spectre ou un pseudo-spectre indiquant selon la direction d'observation l'importance ou non d'une source dans cette direction.

Au nombre de ces algorithmes, il existe la formation de voies [1] qui donne une estimation de la puissance provenant de la direction visée. Par contre, le spectre obtenu est beaucoup moins précis que ceux avec les algorithmes à haute résolution. Le maximum de vraisemblance [1] ou (Maximum likelihood en anglais) est un algorithme à haute résolution basé sur des calculs itératifs de probabilités conditionnelles, mais efficaces dans le cas d'interférences cohérentes. Considéré comme le plus utilisé pour les applications de DOA, l'algorithme MUSIC (pour MUltiple SIgnal Classification) [1] [14] est hautement précis même en présence de bruit. MUSIC exploite les sous-espaces engendrés par les vecteurs propres issus d'une décomposition en amont de la matrice de covariance des signaux reçus. Sa principale faiblesse outre la lourdeur du temps de calcul est la médiocrité de l'estimation en présence d'interférence d'une onde cohérente au signal incident au réseau. Pour corriger cette imperfection, la technique du lissage spatial [11] est souvent jumelée à MUSIC. La liste n'est pas exhaustive, il existe davantage d'algorithmes tels ESPRIT [14], Capon et même d'autres, basées sur les réseaux de neurones idéaux pour les applications temps réel. Ces derniers donnent toutefois des résultats moins précis. Toutes ces méthodes justifient leur existence à travers le compromis entre la précision et la lourdeur du temps de calcul pour des applications nécessitant un traitement en temps réel. Cette dernière limitation à savoir la lourdeur du temps de calcul est de moins en moins limitative de par le développement de circuits intégrés à très haute densité, regroupés sous le sigle de VLSI (Very Large Scale Integration). Ceux-ci sont dotés d'une puissance de calcul incomparable provenant de structures basées sur le parallélisme et permettant des cycles d'opération extrêmement courts.

Par ailleurs, plusieurs défauts d'ordre pratique inhérents aux imperfections de fabrication des antennes et cartes électroniques *RF Front-end* engendrent un effet d'erreur-avalanche depuis les données reçues à l'estimation de la DOA en passant par la structure spécifique de la matrice de covariance. Cette dernière est le cordon ombilical de la quasi-totalité des méthodes de détermination des DOAs de second ordre existantes. La solution idoine et inévitable pour corriger ces erreurs est de calibrer les données reçues afin qu'elles se rapprochent de celles du modèle théorique. On recense dans la littérature plusieurs types d'approches de calibration, regroupés en trois catégories. La première fait fi du problème de calibration en tentant de reconstruire une nouvelle matrice de covariance des données respectant les caractéristiques visées [19]. La seconde approche est connue sous l'appellation d'auto calibration ou de calibration en ligne [3], se traduit par le fait qu'elle ne nécessite pas la connaissance de la position des sources émettrices. Enfin, la troisième appelée calibration supervisée ou différée [4] et [9], nécessite contrairement à la seconde la connaissance de la position des sources servant à effectuer la calibration.

1.1 Objectifs

L'objectif principal du projet vise la mise en place complète d'un traitement numérique sur une puce électronique dédiée, FPGA (Field-Programmable Gate Array) basé sur l'algorithme super résolutif du second ordre MUSIC en vue d'estimer en temps réel la ou les directions d'arrivée des sources de signaux électromagnétiques reçus par un réseau de 8 antennes. Ces signaux émis à des fréquences près de 10 GHz se propageront donc dans l'air pour atteindre les antennes réceptrices, puis traverseront

l'étage RF (Radio Fréquence) ayant pour but de les ramener en bande de base ou à tout le moins abaisser suffisamment leur spectre pour qu'ils soient échantillonnés adéquatement. Le réseau d'antennes et l'étage RF a déjà été réalisée en 2005 et a fait l'objet d'études précédentes dont celle de Hugo Bertrand [5] au sein du Laboratoire de Radiocommunications et de Traitement du Signal. L'acquisition et l'échantillonnage étaient alors effectués avec un système *GageScope 250*, embarqué dans un *PC-486*, qui produisait un fichier de type *.sig* pour chacun des 16 canaux. Le traitement était donc réalisé de manière différée.

Afin de pouvoir obtenir ce traitement en temps réel, les objectifs spécifiques du projet sont :

— Coder l'algorithme de second ordre MUSIC, permettant entre autres d'effectuer l'évaluation de la direction d'arrivée des signaux reçus par le réseau d'antennes, sur la carte FPGA.

— Calibrer les données en vue de compenser les sources d'erreurs qui peuvent survenir dans la réalité expérimentale sur l'amplitude et/ou la phase, sur le couplage mutuel ou sur l'effet des diffuseurs et sur le déséquilibre entre les branches I (Phase) et Q (Quadrature) de chaque antenne. Ces sources d'erreurs induisent une perturbation dans le modèle statistique du signal qui occasionne une mauvaise estimation de la position des signaux émis.

— Remplacer le *PC-486* par une carte d'acquisition de signaux et une carte FPGA afin d'accroitre la performance, d'assurer un traitement temps réel et apporter une plus grande flexibilité future au système.

1.2 Plan du memoire

Ce mémoire commence par une brève définition de certains concepts inhérents au thème du projet, à savoir les modèles du signal et du canal, le concept des réseaux d'antennes, la définition de l'algorithme MUSIC et ses variantes, puis le principe de conception basée sur les modèles et ses dérivés.

Le chapitre 3 : support matériel et logiciel, liste l'ensemble du matériel utilisé en émission comme en réception, les caractéristiques de ceux-ci, leurs fonctions, les réglages appliqués ainsi que leurs branchements. Ensuite, un gros plan est fait sur le calculateur qui sera à programmer. En ce qui a trait à cette étude, il s'agit d'un équipement de la firme *Nutaq*^{TM}, partenaire au projet, connu sous le vocable de *PicoDigitizer*. Dans le même chapitre, les notions sur les logiciels ayant servi d'environnement de développement et le langage de programmation utilisé sont rapportées.

Après avoir pris connaissance du fonctionnement du matériel et du logiciel utiles à l'étude, le chapitre 4 présente les différentes voies possibles de solutions théoriques ou modèles mathématiques proposés dans la littérature pour chacune des étapes de la chaine de réalisation de l'algorithme MUSIC et de la calibration des données. De cette revue de littérature est déduite pour chaque sous-partie du projet une méthode théorique moins demandante en ressources et donnant des résultats acceptables.

Le chapitre 5 présente les détails de la méthodologie adoptée pour l'implémentation sur FPGA du programme général composé des traitements initiaux des données, de la calibration et finalement de

l'algorithme MUSIC. Dans ce chapitre, pour chaque bloc du programme, l'architecture développée est illustrée et analysée, leur fonctionnement et leur rôle sont étayés.

Enfin, le dernier chapitre présente et compare les résultats expérimentaux obtenus en différé sur PC et en temps réel sur FPGA. Avant tout, la méthodologie de déroulement des tests est expliquée. Ensuite, les résultats obtenus sous FPGA à certaines phases du processus du projet - l'obtention de la matrice de covariance, la transformation unitaire, la décomposition en valeurs et vecteurs propres - sont analysés et comparés à ceux obtenus en différé sur PC-*Matlab*.

Chapitre 2

Notions théoriques

2.1 Introduction

Avant d'entamer plus profondément ce mémoire, il est important d'établir une vue d'ensemble sommaire de certaines notions inhérentes à la problématique du projet, d'où l'objet de ce second chapitre. De prime à bord, ce chapitre définit les notations usuelles et conventionnelles en traitement d'antenne. Il fait une présentation générale du réseau d'antennes linéaire uniforme tout en relevant son atout pour l'évaluation de la direction d'arrivée. Ensuite, il aborde la question de la formulation des signaux à la réception du réseau et introduit succinctement les différents algorithmes MUSIC. D'autre part, la définition générale du FPGA est faite ainsi que l'approche adoptée pour sa programmation. Toutes ces informations seront utiles sous une forme ou une autre dans la suite du document.

2.2 Traitement du signal

Soit un système sans fil composé de M sources émettrices dont le signal de la première source est supposé désiré et les $M-1$ autres signaux considérés interférents. Tous ces signaux traversent un milieu, appelé un canal avant d'atteindre N capteurs ou antennes identiques, distribués dans l'espace et formant un réseau. Ce canal est généralement très hostile par nature aux signaux qui s'y propagent en raison de la présence de nombreux diffuseurs ou obstacles avoisinants, causant de multiples réflexions des signaux originaux émis. Ces réflexions s'ajoutent à la superposition des M signaux sources et le bruit gaussien additif créé par l'électronique de réception à chaque capteur. À la réception, au niveau des N capteurs du réseau, il en résulte N copies d'une combinaison de signaux incidents et réfléchis.

Ce système est considéré dans les sections suivantes et prend en compte trois suppositions sans perte de généralité pour simplifier les formulations à venir. La première porte sur l'utilisation d'un réseau d'antennes uniforme et linéaire. La seconde limite les signaux émetteurs comme étant à bande étroite. Enfin, on considère un plan où seul l'angle θ entre l'émetteur et l'axe du réseau est considéré.

2.2.1 Notation et convention

Par souci de clarté, il est bon de souscrire à la notation sous forme matricielle ou vectorielle des différents signaux à traiter. Ces formes doivent respecter les conventions usuelles en traitement d'antennes suivantes :

— Les vecteurs sont notés en **gras** par une lettre minuscule et sauf indication spécifique, sont des vecteurs colonnes.

— Les matrices sont notées en **gras** par une lettre Majuscule.

— Les dimensions des matrices et vecteurs respectent la notation dans l'ordre standard par le nombre de lignes suivie du nombre de colonnes.

— Au niveau des opérations sur les vecteurs et les matrices, on note la transposition, la conjugaison et la transposition-conjugaison qui sont représentées respectivement par les symboles $\top$, $*$ et $\dagger$.

Aussi, un autre terme à considérer représentant les échantillons prélevés à la sortie de chacun des capteurs du réseau à un temps donné $t = kT$ est le vecteur "épreuve" $\mathbf{x}_k$:

$$\mathbf{x}_k = [x_1(k)\, x_2(k) \ldots x_N(k)]^\top \tag{2.1}$$

où l'indice n allant de 1 à N identifie les capteurs du réseau, k l'indice d'échantillonnage et K le nombre total d'échantillons par capteur. Ceci étant, $x_n(k)$, signifie l'échantillon du capteur n enregistré au temps k et le vecteur $\mathbf{x}_k$ de l'équation 2.1 représente l'ensemble des échantillons pour tous les capteurs au temps k. Donc après K épreuves enregistrées pendant une période de temps on obtient une suite de K vecteurs qui concaténés forment une matrice. Dans cette matrice, les colonnes représentent une épreuve $\mathbf{x}_k$.

$$\mathbf{X} = [\mathbf{x}_1\, \mathbf{x}_2 \ldots \mathbf{x}_K]\,. \tag{2.2}$$

En traitement d'antenne, les signaux sont généralement des nombres complexes avec un module et un argument. Ce dernier est primordial pour les signaux à bande étroite autour d'une fréquence porteuse f_0, car il sert de tremplin pour déterminer le déphasage dû à leur retard. Cependant en pratique les signaux réels reçus par le réseau ne sont pas complexes. Il est possible de récupérer le signal analytique (sous forme complexe) avant l'échantillonnage par une démodulation en phase et quadrature.

2.2.2 Réseau d'antennes liniéaire uniforme

Le réseau linéaire uniforme (ULA) est un réseau de N éléments rayonnants alignés et espacés par une distance d identique. La figure 2.1 est un cas général de représentation d'un ULA. En effet, l'axe du réseau n'est pas précisé (x, y ou z), donc l'angle entre la direction d'observation et l'axe du réseau, Ψ est quelconque. Pour simplifier les calculs, l'axe z sera considéré impliquant l'utilisation de l'angle θ.

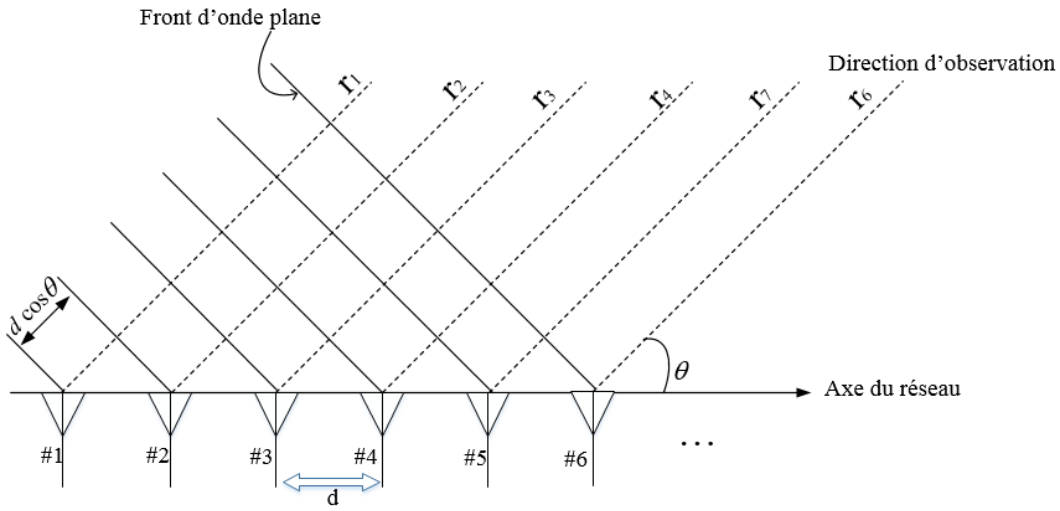


FIGURE 2.1 – Réseau d'antennes linéaire uniforme

En présence d'une source située dans la direction d'observation (angle θ), un front d'onde considéré plan arrive sur le réseau. En fonction de l'angle θ de la source, on observe une différence de parcours de l'onde avant qu'elle parvienne à chacun des capteurs. Cette différence de parcours entre un capteur quelconque n et le capteur 1 pris comme référence peut être exprimée par l'équation suivante :

$$r_1 - r_n = (n-1)d\cos\theta \tag{2.3}$$

où d désigne la distance inter élément et θ l'angle d'incidence du front d'onde plan.

On peut déduire de la précédente équation, le délai en temps entre deux capteurs consécutifs comme suit :

$$\tau = \frac{d\cos\theta}{c} \tag{2.4}$$

où c représente la vitesse de la lumière.

De l'équation précédente on déduit le déphasage entre deux capteurs consécutifs

$$\phi = \tau\omega = \frac{d\cos\theta}{c}\frac{2\pi c}{\lambda} \tag{2.5}$$

avec λ désignant la longueur d'onde.

L'expression (2.5) peut être généralisée en expression (2.6) exprimant ainsi le déphasage pour n'importe quel capteur parmi les N du réseau :

$$\phi_n = \frac{(n-1)2\pi d\cos\theta}{\lambda} \tag{2.6}$$

La clé de la détermination de la provenance des signaux repose sur ce déphasage électrique induit sur les différents capteurs par la source.

En outre, il est bien de souligner que les équations présentées ici sont valides uniquement pour le front d'onde plan. En réalité, le front d'onde produit par une source est sphérique. Plus la source est distante du réseau d'antennes, plus grand est le rayon de cette sphère. Ainsi, lorsque le front d'onde est observé au niveau du réseau d'antennes linéaire uniforme celui-ci parait localement plan. Pour obtenir un front d'onde plan, il faut s'assurer que la source est suffisamment loin du réseau afin que les termes en $\frac{1}{r^2}$ et en $\frac{1}{r^3}$ des expressions des champs électriques et magnétiques issues des équations de Maxwell soient négligeables.

Enfin, il est à noter que la distance inter élément d ne doit pas être plus grande que 0.5λ, afin d'éviter le sous-échantillonnage spatial. Autrement dit, la distance inter élément d constitue l'analogue spatial de la fréquence d'échantillonnage dans le domaine fréquentiel, cette dernière devant respecter le théorème de Nyquist stipulant que cette fréquence d'échantillonnage doit être au moins égale au double de la fréquence la plus élevée présente dans le signal à échantillonner. Donc si on ne respecte pas la limite sur d, on aura un signal échantillonné spatialement qui aura pour effet de créer une ambiguïté sur l'angle d'arrivée des sources.

2.2.3 Formulation des signaux reçus

Soient M sources émettant à diverses positions dans l'espace. Le m-ième signal de la direction θ_m reçu par les N antennes possède une enveloppe complexe $s_m(k)$ à l'instant kT. Chaque signal capté correspond à la superposition des M signaux co-canaux à laquelle s'additionne un bruit additif gaussien issu de l'électronique ou des parasites captés. Les signaux émis sont mutuellement indépendants et indépendants du bruit. Les échantillons du bruit sont gaussiens sur chaque canal I et Q complexes indépendants et de variance σ^2.

Le vecteur canal de la m-ième source, aussi appelé signature spectrale ou encore vecteur de réponse du réseau, regroupe le signal envoyé, le gain complexe entre la m-ième source et la n-ième antenne réceptrice et finalement la réponse des canaux de réception complexe en I et Q de la n-ième antenne au m-ième signal. Les amplitudes des signaux étant similaires sur chacun des capteurs, ils peuvent être simplifiés, car ils ne sont pas affectés par la différence des parcours. Il est alors possible de créer, pour chaque source, un vecteur directionnel $\mathbf{a}$ de la forme suivante :

$$\mathbf{a}(\theta) = \frac{1}{\sqrt{N}}\left[1\, e^{-j\phi(\theta)}\, e^{-j2\phi(\theta)} \cdots e^{-j(N-1)\phi(\theta)}\right]^{\top} \tag{2.7}$$

$$\phi(\theta) = \frac{2\pi d}{\lambda}\cos(\theta). \tag{2.8}$$

Ces vecteurs contiennent l'information nécessaire pour extraire la direction de la provenance des signaux. Ils dépendent de la distance inter capteur d, de la longueur d'onde λ et de l'angle d'observation

θ par le réseau.

Ainsi, une épreuve correspond à l'équation (2.1) et s'exprime en prenant en compte le bruit $\mathbf{n}(k)$ comme suit :

$$\mathbf{x}_k = \sum_{m=1}^{M} s_m(k)\mathbf{a}(\theta_m) + \mathbf{n}(k). \tag{2.9}$$

À un instant d'échantillonnage k, le bruit additif gaussien observé aux N capteurs et l'ensemble des enveloppes complexes des sources sont représentées respectivement par les vecteurs $\mathbf{n}_k$ et $\mathbf{s}_k$. Pour toutes les K épreuves, ces vecteurs forment les matrices $\mathbf{S}$ et $\mathbf{N}$ exprimées par les équations suivantes :

$$\mathbf{s}_k = \left[s_1(k)\, s_2(k) \ldots s_M(k)\right]^\top \tag{2.10}$$
$$\mathbf{S} = \left[\mathbf{s}_1\, \mathbf{s}_2 \ldots \mathbf{s}_K\right] \tag{2.11}$$
$$\mathbf{n}_k = \left[n_1(k)\, n_2(k) \ldots n_N(k)\right]^\top \tag{2.12}$$
$$\mathbf{N} = \left[\mathbf{n}_1\, \mathbf{n}_2 \ldots \mathbf{n}_K\right]. \tag{2.13}$$

De la même manière, l'équation (2.9) est réécrite en exploitant les équations (2.10) et (2.12) par (2.14). Puis en prenant en compte les K épreuves sous forme matricielle par (2.15)

$$\mathbf{x}_k = \left[\mathbf{x}_1\, \mathbf{x}_2 \ldots \mathbf{x}_K\right] = \mathbf{A}\mathbf{s}_k + \mathbf{n}_k \tag{2.14}$$
$$\mathbf{X} = \mathbf{A}\mathbf{S} + \mathbf{N} \tag{2.15}$$

avec $\mathbf{A}$ une matrice représentant tous les vecteurs directionnels exprimés par l'équation suivante :

$$\mathbf{A} = \left[\mathbf{a}_1\, \mathbf{a}_2 \cdots \mathbf{a}_M\right]. \tag{2.16}$$

La matrice d'autocorrélation spatiale $\mathbf{R}_{xx}$ (ou matrice de covariance de signaux) est définie comme une approximation de l'espérance mathématique $\mathrm{E}\{\cdot\}$ d'une épreuve avec elle même. Cette matrice s'exprime ainsi

$$\underbrace{\mathbf{R}_{xx}}_{N\times N} = \mathrm{E}\{\mathbf{x}_k\mathbf{x}_k^\dagger\} \tag{2.17}$$
$$= \mathbf{A}\underbrace{\mathrm{E}\{\mathbf{s}_k\mathbf{s}_k^\dagger\}}_{\mathbf{R}_{ss}}\mathbf{A}^\dagger + \underbrace{\mathrm{E}\{\mathbf{n}_k\mathbf{n}_k^\dagger\}}_{\mathbf{R}_{nn}} \tag{2.18}$$
$$= \underbrace{\mathbf{A}\mathbf{R}_{ss}\mathbf{A}^\dagger}_{\mathbf{R}_{xx-}} + \mathbf{R}_{nn} \tag{2.19}$$

où $\mathbf{R}_{ss}$ et $\mathbf{R}_{nn}$ désignent respectivement la matrice de covariance des sources et du bruit tandis que la matrice $\mathbf{R}_{xx^-}$ représente la matrice de covariance des signaux sans bruit additif. L'approximation de la matrice de covariance devient de plus en plus exacte lorsque le nombre d'épreuves K augmente et permet de réécrire la matrice de covariance comme suit

$$\mathbf{R}_{xx} \approx \tilde{\mathbf{R}}_{xx} = \frac{1}{K}\mathbf{X}\mathbf{X}^{\dagger}. \tag{2.20}$$

2.3 Algorithme de DOA MUSIC

L'algorithme MUSIC, est une méthode de haut niveau d'estimation des DOAs. Il en existe plusieurs variantes aujourd'hui et c'est sans doute la méthode la plus étudiée dans le domaine des DOAs. En outre, MUSIC fait partie intégrante de la famille des méthodes à structures propres. Ces dernières sont axées sur la matrice de covariance, sa décomposition en valeurs et vecteurs propres, plus spécifiquement l'espace engendré par les vecteurs propres. Cet espace peut être partitionné en deux sous-espaces, à savoir sous-espace source et sous-espace bruit. Les méthodes à structure propre tirent avantage de la propriété d'orthogonalité entre ces deux sous-espaces et entre les vecteurs directionnels associés aux sources et le sous-espace bruit pour déterminer les angles d'arrivée des sources en effectuant des projections de vecteurs d'analyse $\mathbf{a}(\theta)$ dont le calcul est similaire aux vecteurs directionnels dans le sous-espace bruit.

2.3.1 MUSIC standard ou Spectral MUSIC

D'abord, le processus de la forme standard de MUSIC débute par la décomposition en valeurs et vecteurs propres (EingenValue Decomposition EVD en anglais), de la matrice de covariance des signaux $\mathbf{R}_{xx}$ de taille $N \times N$. Ensuite, l'espace engendré est scindé en sous-espace source, comprenant les vecteurs propres associés aux M valeurs propres significatives et en sous-espace bruit formé par les vecteurs propres correspondant aux $N-M$ valeurs propres les moins significatives ou nulle dans le cas d'une matrice de covariance idéale (sans buit). Des détails algébriques sur la décomposition EVD peuvent être trouvés dans [4] ; il en résulte les équations suivantes :

$$\mathbf{R}_{\mathbf{xx}} = \mathbf{V}\mathbf{D}\mathbf{V}^{\dagger} \tag{2.21}$$

$$\underset{(N\times N)}{\mathbf{V}} = [\underbrace{\mathbf{v}_1\, \mathbf{v}_2 \cdots \mathbf{v}_M}_{\mathbf{V}_s}\, \underbrace{\mathbf{v}_{M+1} \cdots \mathbf{v}_N}_{\mathbf{V}_n}] \tag{2.22}$$

$$\underset{(N\times N)}{\mathbf{D}} = \mathrm{diag}\{\lambda_1, \lambda_2, \cdots, \lambda_M, \lambda_{M+1}, \cdots, \lambda_N\} \tag{2.23}$$

où $\mathbf{D}$ et $\mathbf{V}$ correspondent respectivement à la matrice des valeurs propres et la matrice des vecteurs propres. $\mathbf{V}_s$ représente le sous-espace source associé aux valeurs propres $\lambda_{1:M}$ en ordre décroissant, $\mathbf{V}_n$ le sous-espace bruit associé aux $\lambda_{M+1:N}$. Ces dernières valeurs propres sont nulles lorsque la matrice

de covariance décomposée est $\mathbf{R}_{xx^-}$. Elles avoisinent les zéros dans le cas de la décomposition de la matrice de covariance $\mathbf{R}_{xx}$ avec bruit.

Une fois le sous-espace bruit trouvé, la recherche des M directions est réalisée en scrutant le ou les vecteurs d'analyse qui seront les plus orthogonaux que possible au sous-espace bruit. Cela est possible grâce au pseudo-spectre de puissance MUSIC exprimé par l'équation (2.24) en faisant varier θ.

$$\mathbf{P}_{mus}(\theta) = \frac{1}{\mathbf{a}^{\dagger}(\theta)\mathbf{P}_n\mathbf{a}(\theta)} \tag{2.24}$$

$$\mathbf{P}_n = \mathbf{V}_n\mathbf{V}_n^{\dagger} = \mathbf{I} - \mathbf{V}_s\mathbf{V}_s^{\dagger} \tag{2.25}$$

Le dénominateur représente la projection des vecteurs de balayage sur le sous-espace bruit, ce qui est similaire à la distance euclidienne entre le vecteur de balayage et le sous-espace bruit. Lorsque cette distance est nulle, ou presque, cela signifie que l'angle pour lequel on obtient cette projection est la direction recherchée d'une source.

En général, le pseudo-spectre de MUSIC est tracé en fonction de θ en dB, offrant une identification des directions d'arrivée par observation des maximas locaux qui surviennent à la division par distance euclidienne quasi nulle. Toutefois, il n'est pas un bon indicateur si l'on veut obtenir le niveau de la puissance des signaux, à cause de la relation d'inversion.

Enfin, l'algorithme MUSIC nécessite la connaissance du nombre de sources M pour la partition de l'espace en sous-espaces bruit et source. Il existe plusieurs méthodes à cet effet : celles basées sur l'estimation de l'ordre, telles que Akaike Information Criteria (AIC) ou Minimum Descriptive Length, ou bien la méthode utilisant le critère de l'erreur quadratique moyenne RMSE, basée sur le pourcentage fixé par l'usager de la somme des valeurs propres pour en dénombrer les plus significatives. En général, le pourcentage RMSE permis est de 5.

2.3.2 ROOT MUSIC

Une variante de la procédure MUSIC standard définie ci-dessus dans le cadre d'une ULA, se base sur la recherche des racines d'un polynôme pour déterminer les DOAs. Elle est connue sous l'appellation de ROOT-MUSIC ou la version Racine de MUSIC. Contrairement à la technique standard de MUSIC basée sur l'identification et l'observation des pics grâce au traçage du graphique du pseudo-spectre, ROOT MUSIC permet d'obtenir directement les DOAs assimilées aux racines d'un polynôme.

On peut écrire l'équation du polynôme comme suit :

$$Q(z) = \sum_{n=-(N-1)}^{N-1} p_n z^{-n} \quad \text{avec} \quad z^{-n} = e^{jn\phi} \tag{2.26}$$

où p_n représente la somme des éléments de la n-ième diagonale de la matrice de projection $\mathbf{P}_n$ (2.24), z un déphasage produit par un retard avec n l'indice de l'élément de l'antenne réseau, ϕ_m les déphasages inter éléments reliés à la position des sources. Les solutions de ce polynôme sont des racines doubles z_m à module unitaire, dont l'argument procure les valeurs des ϕ_m. Ainsi, l'angle d'arrivée θ_m peut être déduite en utilisant une des racines doubles ayant un module proche de l'unité et en développant l'expression de ϕ_m, comme le montre l'équation (2.27) en utilisant l'équation (2.5)

$$\hat{\theta}_m = \arccos\left(\frac{-\arg(\mathbf{z}_m)}{2\pi\frac{d}{\lambda}}\right) \tag{2.27}$$

où $\arg(\cdot)$ signifie argument.

2.3.3 Le lissage spatial

La méthode de MUSIC standard, lorsqu'elle est utilisée dans le cadre d'une estimation de la direction d'arrivée des signaux provenant de sources corrélées, atteint ses limites. C'est pourquoi elle est parfois couplée à des techniques comblant ce déficit, entre autres le principe du lissage spatial. Ce concept de lissage spatial permet de diminuer la corrélation entre les sources en déterminant une nouvelle matrice $\mathbf{R}'_{xx}$ de covariance. Cette matrice est obtenue par un moyennage d'un ensemble de matrices partielles formées par des sous réseaux dont le nombre d'éléments est inférieur à N le nombre de capteurs du réseau complet. La figure 2.2 illustre cette partition du réseau linéaire de N éléments en N_r sous-réseaux de $N_p = N - N_r + 1$ éléments.

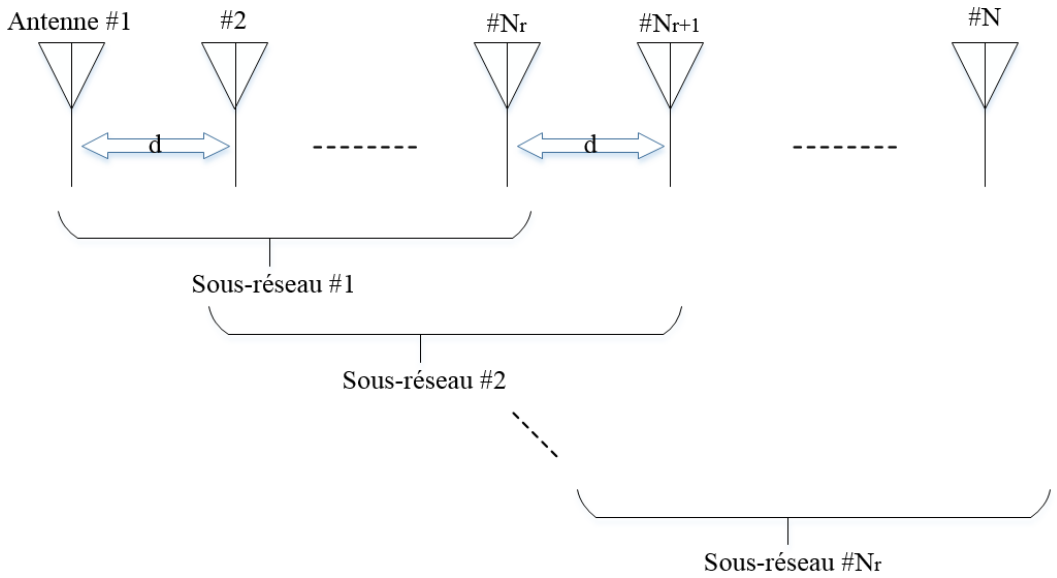


FIGURE 2.2 – Décomposition du réseau principal en sous-réseaux

Comme chaque sous-réseau induit une matrice partielle $\mathbf{R}_p(k)$ comprise dans la matrice de covariance globale originale $\mathbf{R}_{xx}$, on peut exprimer mathématiquement le lissage spatial par l'équation (2.28)

$$\mathbf{R}'_{xx} = \frac{1}{N_r} \sum_{k=1}^{N_r} \mathbf{R}_p(k) \tag{2.28}$$

Par ailleurs, la technique du lissage spatial omet de prendre en compte certains éléments de la matrice de covariance originale, ceci a un impact sur la qualité des résultats des techniques spatiales utilisant cette matrice de covariance lissée. Ainsi, pour pallier ce problème, la méthode de lissage bidirectionnel vient améliorer la technique standard en réduisant davantage le facteur de corrélation entre les sources.

Le lissage spatial bidirectionnel permet d'obtenir une nouvelle matrice de covariance $\mathbf{R}_{finale}$ composée de la matrice de corrélation obtenue par la technique du lissage spatial $\mathbf{R}'_{xx}$ et une seconde matrice $\mathbf{R}''_{xx}$ de covariance obtenue en multipliant la matrice $\mathbf{R}^{\dagger}_{xx}$ à gauche et à droite par la matrice anti-diagonale $\mathbf{J}$.

$$\mathbf{R}''_{xx} = \mathbf{J}(\mathbf{R}'_{xx})^{\dagger}\mathbf{J} \tag{2.29}$$

$$\mathbf{R}_{finale} = \frac{1}{2}\left(\mathbf{R}'_{xx} + \mathbf{R}''_{xx}\right) \tag{2.30}$$

2.4 Field Programmable Gate Array (FPGA)

Un FPGA est un dispositif en silicium contenant des cellules logiques conçues par un manufacturier en vue d'être reprogrammées à souhait par un usager-concepteur. Ces cellules logiques programmables sont connues sous le sigle CBL pour Configuration Logic Block. Chacune d'entre elles contient dépendamment de la version du dispositif FPGA, plus ou moins de Look Up Table (LUT), de multiplexeurs et bascules. Entre les CBLs, il existe des ressources de routage ou d'interconnexions et ils sont bordés par des ports d'entrée-sortie.

En général, un FPGA est programmé avec un langage de description matériel (Hardware Description Language : HDL) comme le VHDL ou Verilog. Il est souvent utilisé pour des applications requérant une exécution en temps réel, grâce à son architecture allouant un traitement des tâches en parallèle. La figure 2.3 illustre les grandes étapes de programmation d'un FPGA.

2.5 Conception basée sur les modèles (MBD)

La conception basée sur les modèles (MBD) est intéressante lors de la mise en œuvre de systèmes complexes à grande échelle (de grande taille). C'est un processus permettant le développement rapide des systèmes dynamiques, des systèmes de contrôle, de traitement de signal et des systèmes de communication. Son avantage principal réside dans le mot *modèle*. En effet, le MBD traduit les spécifications du système à concevoir en un ensemble de blocs et sous-blocs structurés facilitant sa lecture et sa réalisation. Le modèle peut être simulé et testé tout au long du processus de développement. À la fin de la conception, le modèle fonctionnel retenu peut être utilisé pour générer automatiquement un code dans un langage informatique souhaité par l'usager. La figure 2.4 résume le concept du MBD.

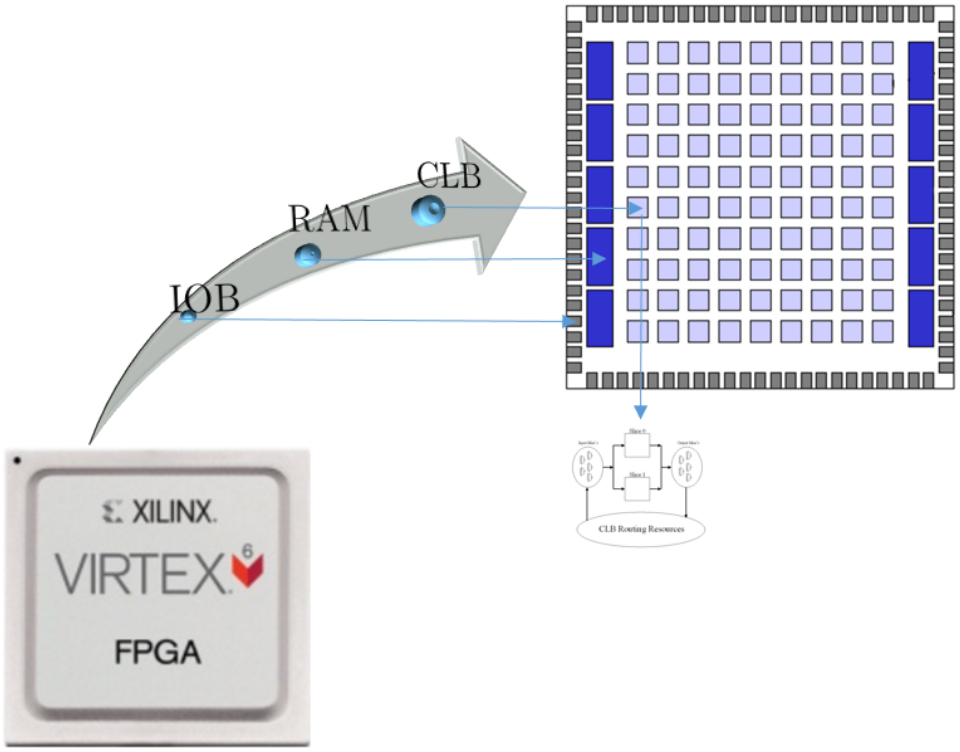


FIGURE 2.3 – Un FPGA

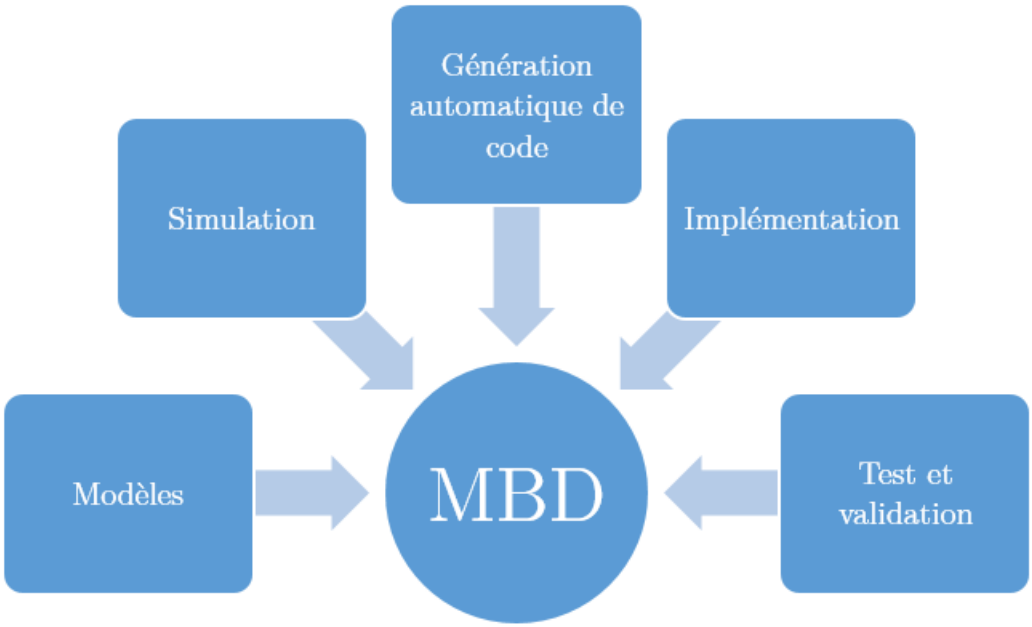


FIGURE 2.4 – Diagramme de la conception basée sur les modèles

2.6 Concept du prototypage rapide des lois de commande

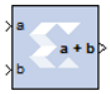
En allant plus vers le matériel, le concept MBD est souvent désigné comme le concept de prototypage rapide (RCP). Il permet tester et développer des systèmes assez complexes en temps réel d'une manière rapide et facile. Les deux attributs principaux de la RCP sont : la programmation visuelle de haut niveau et la génération automatique de code de bas niveau (langage proche du matériel).

La différence entre la mise en œuvre FPGA via un programme bas niveau écrit entièrement en VHDL et celle via un programme issu de la génération automatique de code basée sur la modélisation haut niveau est associée au temps de réalisation, à la flexibilité et à la connaissance. En effet, il est possible de simuler, tester, valider des algorithmes et les implémenter dans un matériel sans une profonde connaissance des HDLs comme VHDL ou Verilog. En exemple, la figure 2.5 montre comment une simple addition entre deux valeurs est implémentée en VHDL (code bas niveau) et comment elle est modélisée dans un langage haut niveau.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Adder is
   port
   (
      a, b : in unsigned(3 downto 0);
      sum       : out unsigned(3 downto 0);
      carry_out : out std_logic
   );
end entity Adder;

architecture Behavioral of Adder is
   signal temp : unsigned(4 downto 0);
begin
   temp <= ("0" & a) + b;
   sum       <= temp(3 downto 0);
   carry_out <= temp(4);
end architecture Behavioral;
```



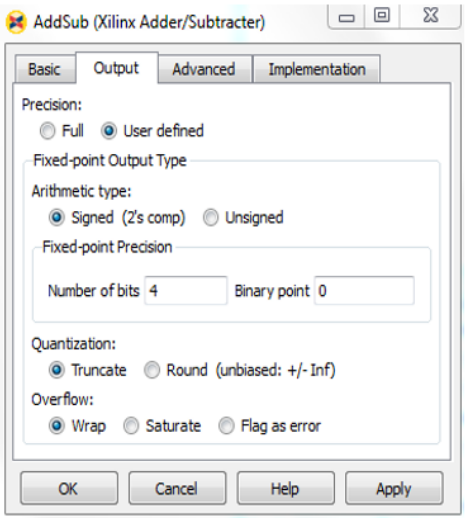


FIGURE 2.5 – À gauche, un additionneur écrit en VHDL et à droite un bloc additionneur avec sa fenêtre de dialogue

2.7 Conclusion

Les différentes notations et conventions en traitement d'antenne qui seront adoptées dans les prochains chapitres ont été présentées. Ces notations représentent les signaux formés par le réseau d'antennes. Ces signaux sont sous forme de vecteurs et de matrices, correspondant aux échantillons reçus. La formulation des signaux reçus par le réseau d'antennes a permis d'introduire les vecteurs directionnels et la matrice de covariance. Cette dernière est utilisée par les algorithmes de DOA pour extraire les angles d'arrivée.

Ensuite, un gros plan a été fait sur l'algorithme MUSIC et ses variantes. La version standard ou spectrale MUSIC dont la résultante est un graphique indiquant les angles d'arrivée (pseudo-spectre) a

d'abord été abordée. Après cela, l'autre variante Root-MUSIC basée sur les racines d'un polynôme pour trouver les angles d'arrivée a été présentée. Enfin, le concept de lissage spatial a été introduit. Son avantage est d'augmenter le pouvoir séparateur de MUSIC en présence de sources corrélées. Le FPGA, l'élément de traitement a été brièvement défini. Puis, l'approche de programmation basée sur le modèle a été présentée ainsi que le concept de prototypage rapide.

Chapitre 3

Matériel et support logiciel

3.1 Introduction

Le défi du projet relaté en introduction est l'implémentation de MUSIC sur une puce électronique dédiée FPGA en vue de déterminer en temps réel la ou les directions d'arrivée de signaux électromagnétiques à 10 GHz reçus par un réseau de 8 antennes. Ce chapitre présente les différents matériels et logiciels utilisés pour relever ce défi. Premièrement, il met en évidence le montage expérimental au complet aussi bien en émission qu'en réception (créant un environnement de test). Ensuite, un gros plan est fait sur l'unité intelligente, le *PicoDigitizer* permettant l'acquisition et le traitement numérique des signaux. Enfin, ce chapitre apporte des précisions sur les logiciels et le langage utilisés pour la réalisation du projet.

3.2 Fonctionnement général

La figure 3.1 dresse un schéma du fonctionnement général du système. Dans un premier temps une ou des sources émettent un signal autour de 10 GHz se propageant sous forme d'onde électromagnétique dans l'air en direction du réseau de huit antennes cornets. Une fois captée, l'onde pénètre dans un guide d'onde qui est rattaché à l'embout de chacune des antennes. L'onde électromagnétique induit un courant dans le câble entre le guide d'onde et la carte front-end RF. Ensuite, au niveau de la carte RF, les signaux reçus par les 8 capteurs sont amplifiés et mélangés deux fois afin d'augmenter la puissance, d'en abaisser suffisamment la fréquence pour pouvoir les échantillonner. Pour chaque antenne, le signal reçu est décomposé en phase I et quadrature Q avant d'être échantillonné.

Le signal une fois converti de l'analogique au numérique puis échantillonné est traité par un FPGA en temps réel pour calibrer les données et déterminer le pseudo spectre de ou des sources. Finalement, le FPGA transmet par Ethernet les résultats à un PC pour affichage à l'utilisateur.

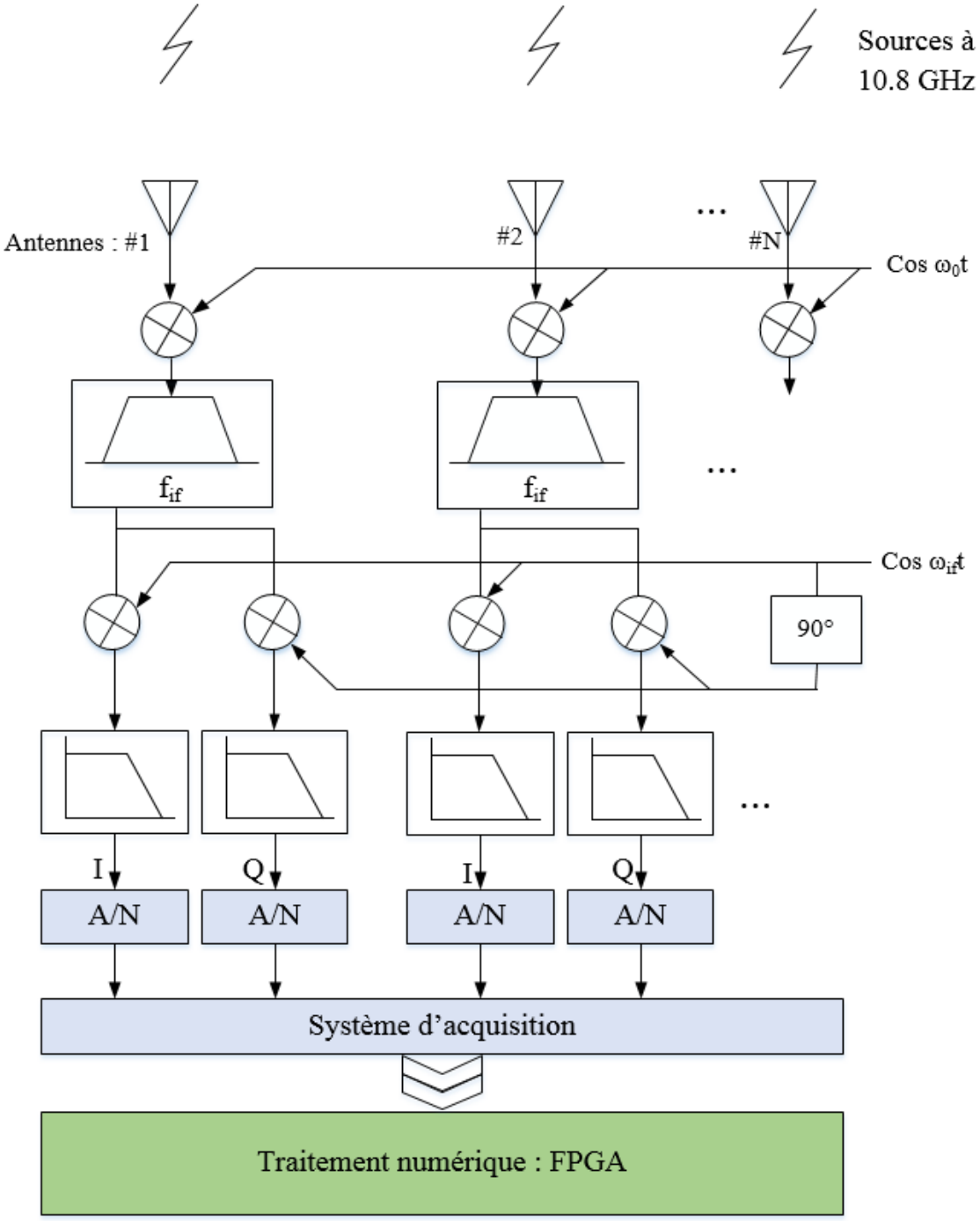


FIGURE 3.1 – Schéma du fonctionnement général

3.2.1 Montage à l'émission

La figure 3.2 illustre le montage complet utilisant deux antennes à l'émission. Ce montage est composé de deux sources, chacune émettant un signal monochromatique pouvant aller au-delà de 10 GHz . Chacune des sources est reliée à une antenne cornet par un câble SMA-SMA. Les deux sources utilisées

sont les modèles 8673C et 83642A de la compagnie HP (Hewlett-Packard).

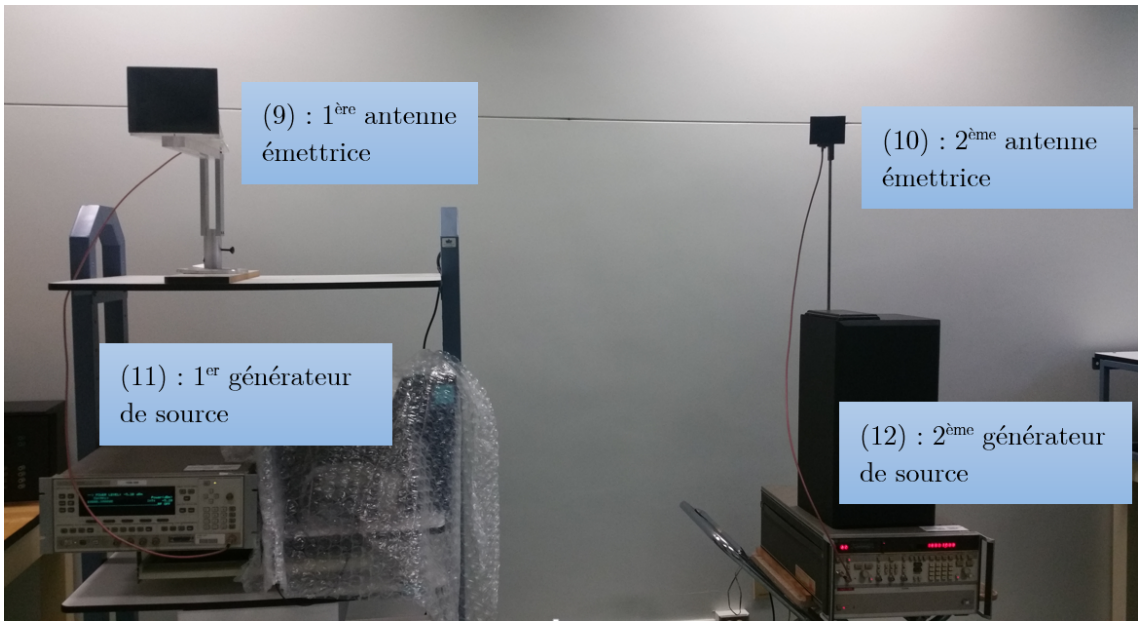


FIGURE 3.2 – Montage d'une antenne à l'émission. Les annotations entre parenthèses sont expliquées à l'annexe A.

3.2.2 Montage à la réception

Le montage global à la réception est illustré par la figure 3.3. Il est constitué d'un réseau de 8 antennes (identifié par 1 sur la figure) supporté par un mécanisme amovible permettant une orientation horizontale (selon l'angle θ). Le montage comprend aussi une carte électronique pour la démodulation (2), de deux générateurs de signaux identifiés sur l'image par (3 et 8), d'un amplificateur RF (7) capable de délivrer une puissance élevée jusqu'à 10 dBm, d'une source d'alimentation de l'électronique (6), d'une unité d'acquisition et de traitement numérique (4) et d'un analyseur de spectre (5). Le dernier item est optionnel, il permet de visualiser le spectre des signaux émis et les valider avant leur traitement.

La figure 3.4 dresse le schéma de la chaîne de réception pour une antenne du réseau (1) jusqu'à l'obtention de composantes en phase et en quadrature. Une fois le signal à 10.8 GHz reçu, il est mélangé (2a) une première fois avec un signal de référence à 10.53 GHz généré par (8) et amplifié par (7) afin d'attaquer convenablement l'entrée LO (oscillateur local) des 8 mélangeurs. Le signal mélangé est ensuite amplifié par (2b), divisé par (2c) en deux afin d'être de nouveau appliqué à un deuxième étage de mélangeurs (2d) avec un signal de référence variable (3) autour de 272 MHz amplifié par (2b) et passé dans un coupleur hybride pour sortir un signal direct et un signal déphasé de 90° par (2e). Ce second étage de mélangeur sert à obtenir les composantes en phase et quadrature du signal. Enfin, la fréquence du signal obtenu est suffisamment abaissée pour que le signal puisse être échantillonné et traité par le FPGA monté dans le *PicoDigitizer* (4). Le tableau récapitulatif des éléments utilisés à la réception ainsi que leur configuration peut être consulté à l'annexe A.

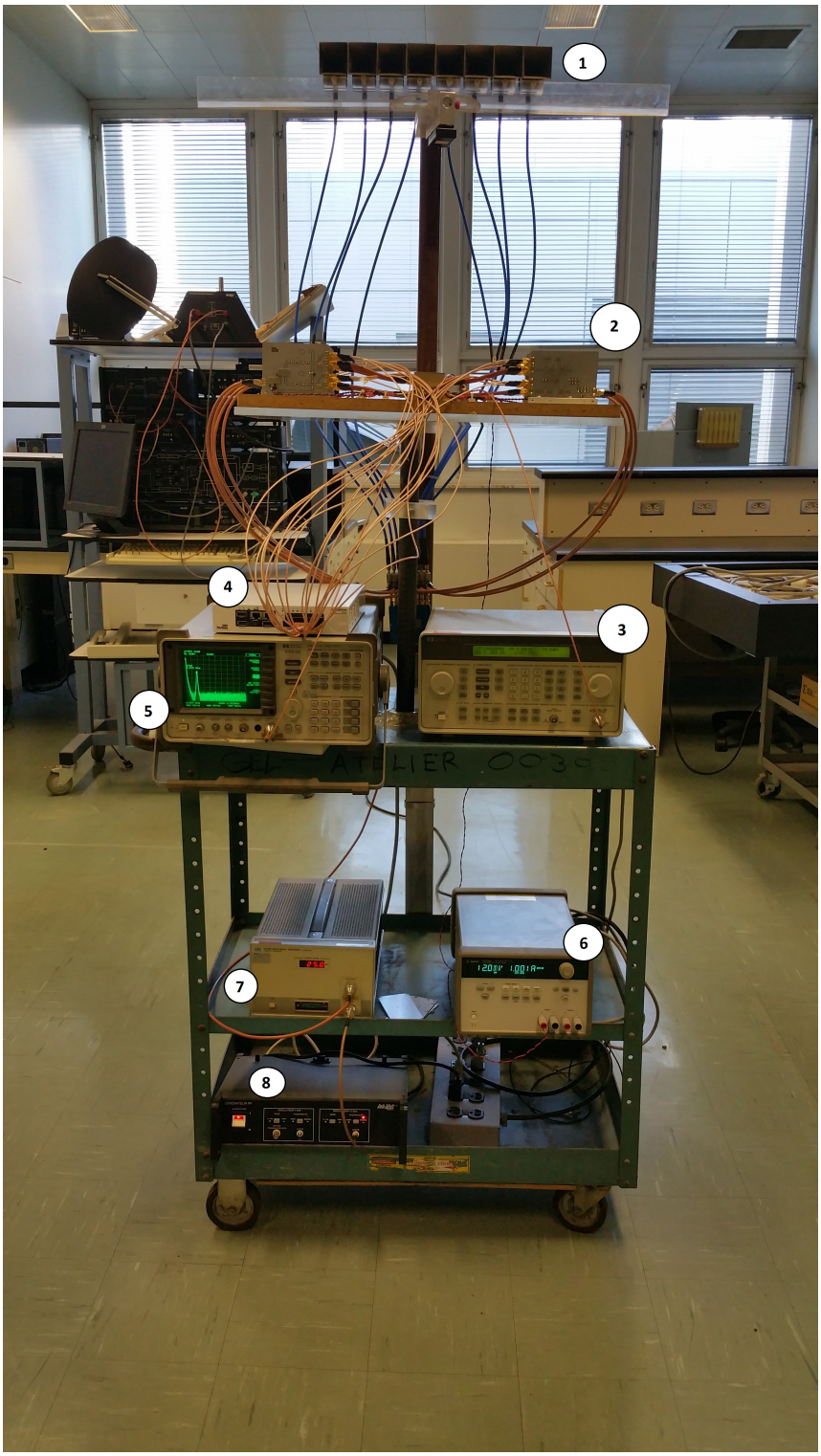


FIGURE 3.3 – Montage global à la réception. Les annotations encerclées 1 à 8 sont expliquées ci-dessus.

N. B. Les valeurs des différentes fréquences précisées dans le tableau et sur le schéma ne sont pas absolues. Elles ont été modifiées tout au long de l'expérimentation.

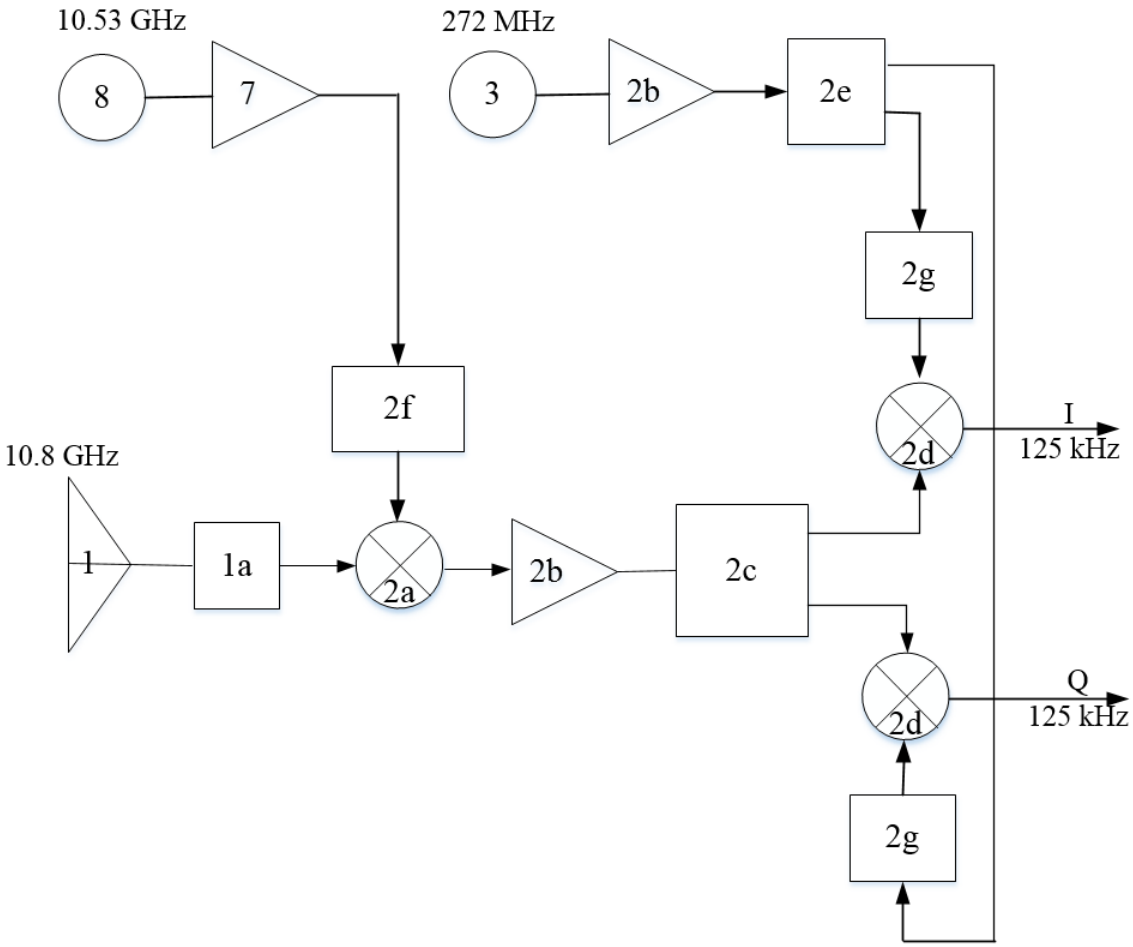


FIGURE 3.4 – Schéma du montage pour une antenne

3.3 Le *PicoDigitizer* (*Nutaq*)

L'élément clé de l'ensemble du matériel utilisé est bien le *PicoDigitizer*. En effet, il permet d'effectuer les opérations d'acquisition des signaux, de conversion de l'analogique au numérique et vice-versa, ainsi que le traitement numérique des algorithmes codés. Le *PicoDigitizer* est une solution DAQ (Data AcQuisition) produite par la compagnie *Nutaq*™. Le modèle utilisé dans le cadre du projet est la série-125 du *PicoDigitizer* non-embarqué. Ce modèle est composé d'une carte d'acquisition (FMC MI-125-16) de 16 canaux A/D extensible (à 32 canaux) avec une résolution de 14 bits et une fréquence d'échantillonnage de 125 Ms/s sur chaque canal. Cette carte est reliée à un FPGA modèle LX240T de la famille des *Virtex-6* de la compagnie *Xilinx*™ainsi qu'une mémoire RAM de 4 GB. Le modèle offre deux ports de communications avec un ordinateur à savoir Giga Ethernet et PCIe. La figure 3.5 prise sur le site internet de *Nutaq* montre les composantes et leur branchement.

Le *PicoDigitizer* couplé aux outils de développement de logiciel élaboré par *Nutaq* présente de nombreux avantages. Entre autres, il y a la réduction du temps de réalisation des applications. En effet, grâce aux boites à outils élaborés par *Nutaq*, certaines tâches fastidieuses courantes telles que

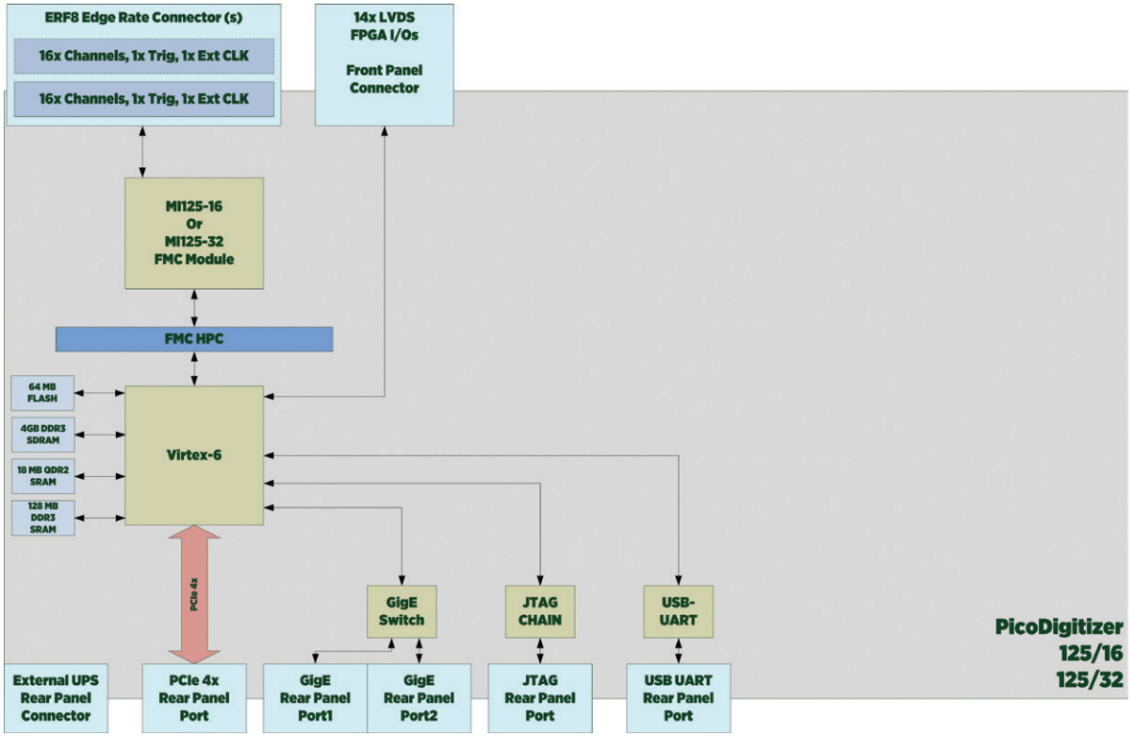


FIGURE 3.5 – Diagramme bloc du *PicoDigitizer* série 125 en version non embarquée (`www.nutaq.com` [consulté le 8 février 2016])

l'interfaçage entre les différentes cartes, l'ajustement des contraintes FPGA, le débogage des pilotes pour la communication avec un hôte, et autres tâches connexes sont accomplies de manière transparente. Aussi, les outils de développement logiciel de *Nutaq* sont fournis avec trois niveaux d'intégration assurant ainsi une grande flexibilité dans le codage des applications. Premièrement, il y a le MBDK pour Model-Based Design Kit qui permet un développement haut niveau d'applications avec *Simulink* et *Xilinx System generator*. Ensuite, il y a le niveau intermédiaire, Board Software Developement Kit (BSDK) et enfin la couche physique Board Support Package (BSP).

Dans le cadre du projet l'outil MDBK a été favorisé afin d'assurer un développement rapide sur FPGA.

3.4 Support Logiciel

Les logiciels à utiliser pour le ciblage et la programmation de la *PicoDigitizer* ou plus précisément la carte *Perseus* porteuse de la carte FPGA dépendent des trois approches MDBK, BSDK, et BSP. Comme, l'approche MBDK est celle adoptée, seulement les outils et logiciels s'y référents sont présentés. Puisque cette approche s'appuie sur le générateur automatique de code de *Xilinx* (*Sysgen* ou *System generator*) pour convertir le code graphique de *Simulink* en fichier binaire pour FPGA, la section suivante introduit cet outil de *Xilinx* avant de définir le système MBDK et les logiciels l'accompagnant.

3.4.1 L'environnement de développement *Simulink*

Simulink est un produit de *Mathworks* de la Compagnie *Matlab*™. C'est environnement de développement graphique destiné à la conception basée sur les modèles. Ainsi, il supporte tous les attributs de la MBD, la simulation, la génération automatique de code C, C++ et HDL, et les tests et vérifications de systèmes embarqués. Il facilite aussi la connexion du modèle final au matériel pour un prototypage rapide et pour la simulation HIL vue au chapitre précédent.

D'autre part, *Simulink* possède une bibliothèque prédéfinie enrichie par la modélisation, sous forme de blocs, d'une grande partie des fonctions de *Matlab*. Cela simplifie la conception de systèmes complexes que ce soit dans le domaine électrique, mécanique ou hydraulique. De plus, ses concepteurs ont tiré avantage de son association avec *Matlab* en permettant l'incorporation de code *Matlab* dans les modèles *Simulink* et en rendant possible l'exportation des résultats de simulation vers *Matlab* pour des traitements plus poussés.

L'une des fonctionnalités intéressantes de *Simulink* est qu'il est extensible. Il permet ainsi d'ajouter à sa bibliothèque des sous-bibliothèques développées par des particuliers ou des entreprises. C'est d'ailleurs cette flexibilité qu'utilise *Xilinx* pour ajouter les blocs reliés à leur produit en associant l'outil *System generator* à *Simulink*. De même, les blocs permettant l'indexation des produits de *Nutaq* sont ajoutés à la bibliothèque de *Simulink* grâce à l'outil MDBK. Les sous-sections suivantes présentent ces deux outils utilisés dans cadre de ce projet. La version *Matlab*/*Simulink* utilisée dans le projet est la R2009b.

3.4.2 *System generator* de *Xilinx*

Un FPGA est configuré en chargeant dans une mémoire statique à accès aléatoire un fichier appelé *bitstream*. Ce *bitstream* est produit par des outils de compilation qui convertissent un modèle image haut niveau d'un programmeur FPGA en un modèle bas-niveau équivalent et exécutable. Faisant partie de ces outils de compilation, le *System generator*, une idée de *Xilinx*, permet de compiler un modèle de haut niveau *Simulink* en un programme FPGA. La version utilisée dans ce projet est 13.4. Une vue d'ensemble du *System generator* est illustrée par la figure 3.6 et les portions identifiées sur la figure sont détaillées ci-après :

1. **Le gestionaire de bibliothèque *Simulink*** - En plus des blocs de *Simulink*, l'ensemble des blocs de *Xilinx* y est listé. Ces blocs comprennent des éléments basiques tels que l'additionneur, multiplexeur, constante, etc... des éléments spécifiques tels que *Blackbox*, le bloc *Cordic* ainsi que d'autres permettant de faire appel à des interfaces à d'autres outils logiciels de *Xilinx* : *FDATool*, *ModelSim*.
2. **Le bloc *System generator*** - Ce bloc doit paraître au moins une fois dans tous modèles *Simulink* comprenant des blocs de la bibliothèque de *Xilinx*. Il est utilisé comme panneau de configuration des paramètres de simulation et matériel (FPGA) à cibler pour la co-simulation, aussi il permet d'invoquer le générateur de code pour la création d'un fichier *bitstream* ou HDL Netlist. Les

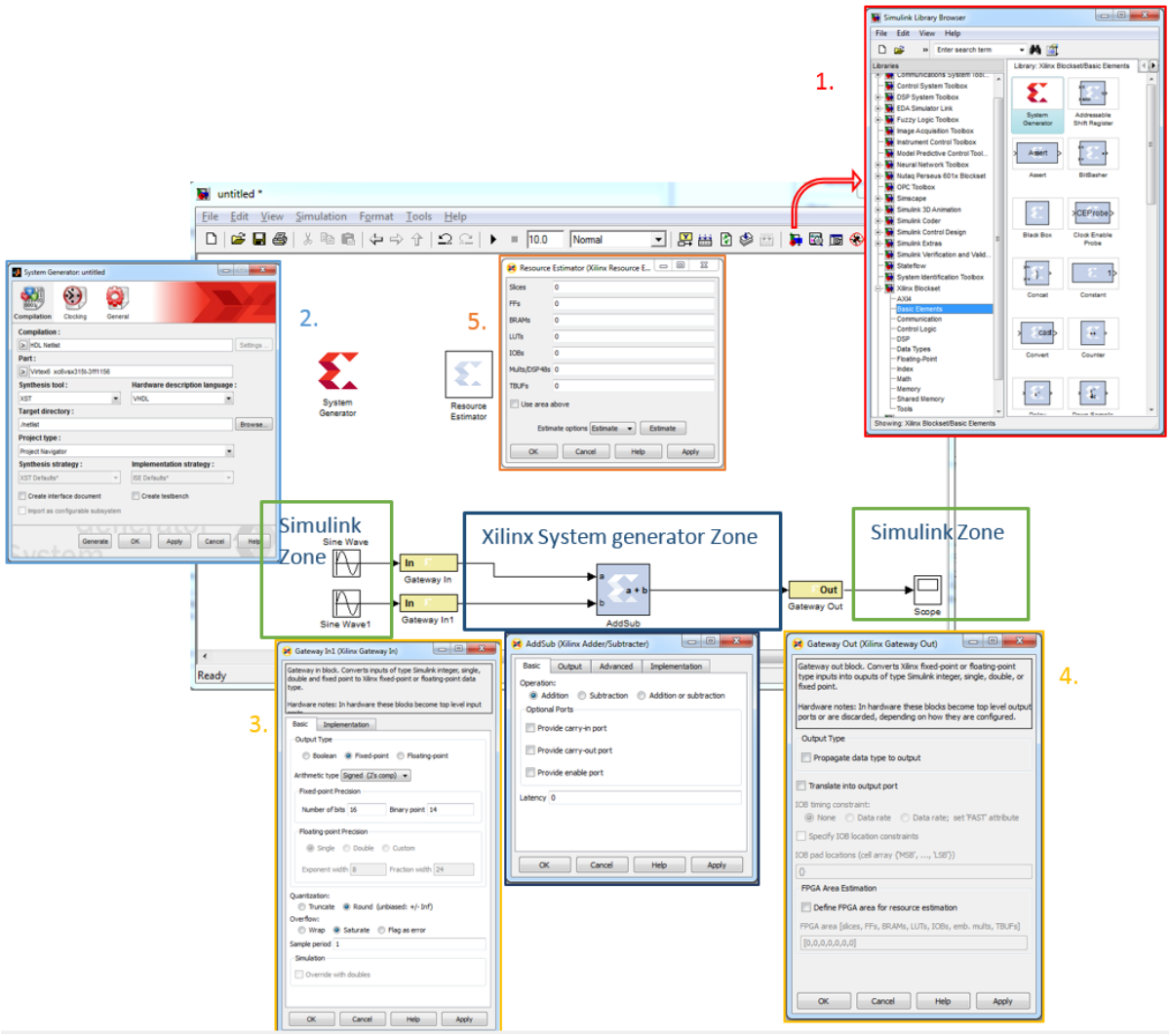


FIGURE 3.6 – Synoptique d'un simple modèle du *System generator* de *Xilinx*

réglages des horloges FPGA et la durée d'une période de simulation depuis l'environnement de développement de *Simulink* sont fixés à partir de ce bloc.

3. **Le bloc Gateway In** - Il délimite la frontière entre les blocs *Simulink* et les blocs de *Xilinx* et précède tout bloc *Xilinx*. Ce bloc en plus de fixer le temps d'échantillonnage, convertit les types de données entier, double, point fixe et virgule flottante de *Simulink* en type de données point fixe de *System generator*.
4. **Le bloc Gateway Out** - Ce bloc convertit le type des données point fixe en provenance des blocs *Xilinx* en type de données (entier, double, point fixe...) destinées aux blocs de *Simulink*. Il est la passerelle pour passer des blocs *Xilinx* aux blocs *Simulink*.
5. **Le bloc d'estimation des ressources** - Il permet une estimation rapide des ressources (slices, bascules, bloc de mémoire RAM BRAM, les blocs d'entrée/sortie et autres) requis pour l'implémentation FPGA du modèle *Simulink*.

Un modèle *System generator* est composé uniquement des blocs *Xilinx* présents dans la sous bibliothèque rajoutée à la suite de l'installation du *System generator*. Lors de la création du fichier *bitstream* le générateur de code prendra en compte seulement les blocs *Xilinx* ; les blocs de *Simulink* seront négligés. Ces derniers sont utilisés en général pour émuler une source ou pour visualiser les signaux durant l'étape de la simulation. Il est important de mentionner certains blocs spéciaux du *System generator* utilisés dans la conception de ce projet. Il s'agit des blocs *MCode* et *Cordic* définis brièvement ci-après. Plus d'informations sur ces blocs peuvent être trouvées respectivement dans [16] et [17].

Le bloc *MCode* et *BlackBox*

Les blocs *MCode* et *BlackBox* sont des conteneurs permettant d'exécuter des fonctions existantes ou codées par l'utilisateur final respectivement en *Matlab* et en VHDL/Verilog au sein de *Simulink*. La figure 3.7 représente le bloc *MCode*. Les paramètres du bloc (entrée, sortie et nom du bloc) sont importés de la fonction *Matlab*. Le bloc par défaut disponible est *xlmax*. Il permet de retourner tout simplement la valeur maximale entre deux valeurs présentées à son entrée. Lors de la simulation, la fonction *Matlab* chargée dans le bloc *MCode* est exécutée et les résultats calculés sont présentés à la sortie du bloc.

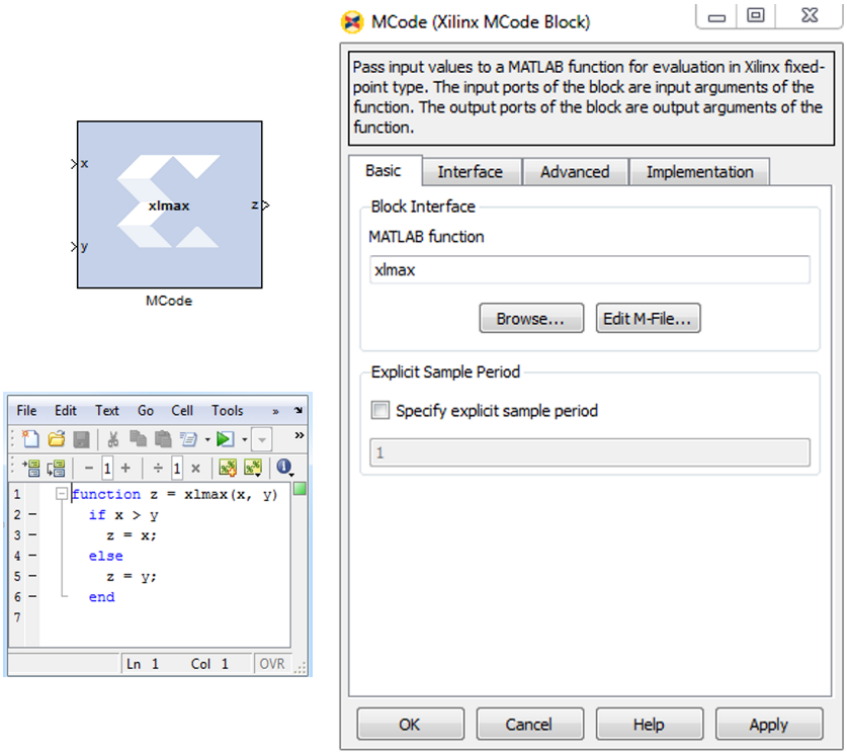


FIGURE 3.7 – Le bloc Mcode, sa boite de dialogue et le code *Matlab*

Le *MCode* supporte un sous-ensemble limité de commandes *Matlab* suffisant pour les fonctions arithmétiques, les machines à état et la logique de commande. De plus, trois règles de conception doivent être respectées à l'écriture du code : toutes les entrées et sorties doivent être typées en *Xilinx*

point fixe, le bloc doit avoir au moins une sortie et enfin le code *Matlab* chargé dans le bloc doit être dans le même répertoire que le modèle utilisant ce bloc. Dans le cadre de ce projet le bloc *MCode* a été utilisé en grande partie comme machine à état.

Le bloc *Cordic*

Le COordinate Rotational DIgital Computer ou *Cordic* est un algorithme qui permet d'implémenter des équations mathématiques basées sur le calcul numérique par rotation de coordonnées. Le bloc *Cordic* 5.0 de *Xilinx* calcule les équations de rotation, translation, sinus et cosinus, sinus hyperbolique et cosinus hyperbolique, arc tangente, arc tangente hyperbolique et racine carrée. Dans le cadre du projet, le *Cordic* réalise les calculs de rotation, tagente et sinus et cosinus. Les paramètres du bloc varient selon la fonction réalisée par celui-ci. La figure 3.8 montre les différentes formes du bloc en fonction du type de calcul sélectionné.

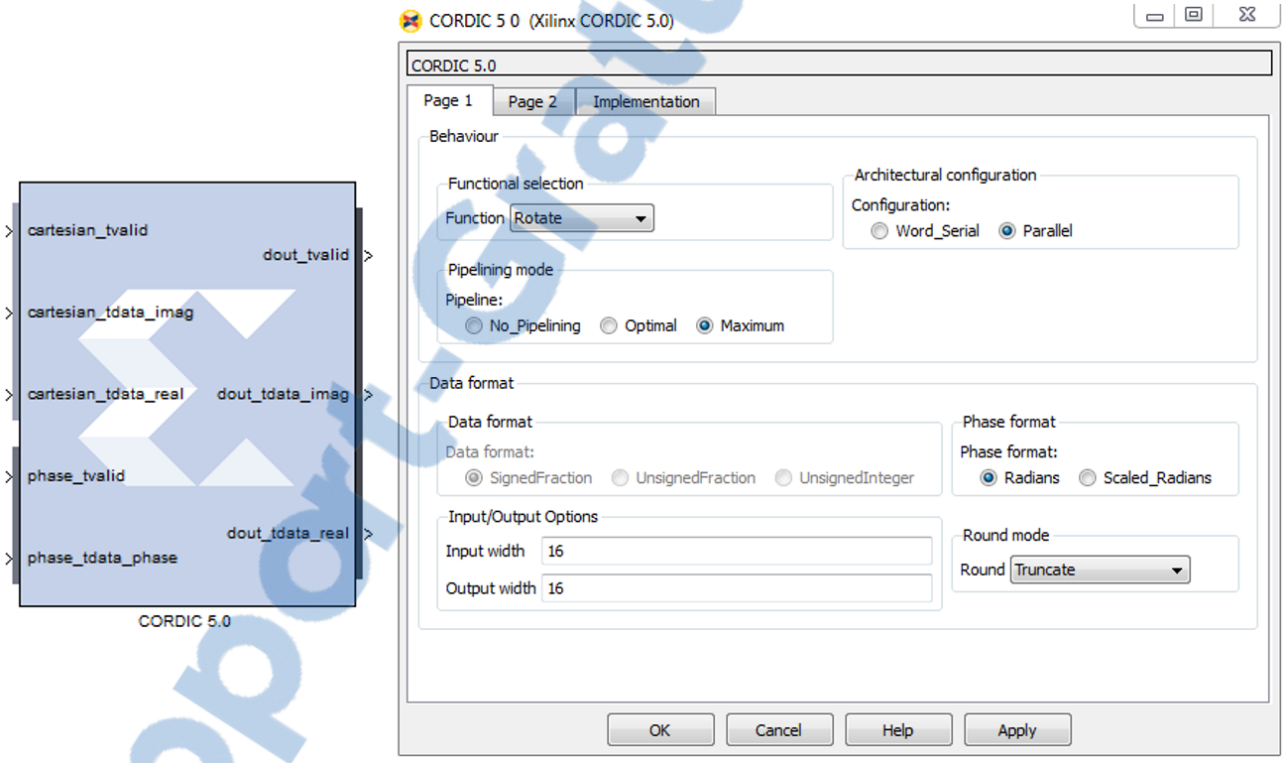


FIGURE 3.8 – Le bloc *Cordic* 5.0 et sa boite de dialogue

3.4.3 MBDK *Nutaq*

Nutaq s'inscrit dans l'approche de conception basée sur les modèles avec son outil de développement MBDK afin de simplifier la tâche aux utilisateurs de leurs produits. Cet outil est en fait une bibliothèque de blocs modélisant des fonctions spécifiques dédiées aux produits de la compagnie.

L'outil MBDK est associé à *Matlab* suite à l'installation du logiciel *ADP6-MicroTCA* de *Nutaq*. La version de ce dernier installée dans le cadre du projet est l'édition 6.6.0.

Structurellement parlant, chaque bloc de la bibliothèque MDBK est un ensemble organisé de code de bas niveau et de haut niveau transparent à l'usager. Par exemple, certains blocs contiennent du code HDL pour échanger des données entre les différentes pièces entourant le FPGA comme la mémoire DDR3, la carte FMC, Gigabit Ethernet, etc. mais, il existe aussi d'autres blocs du MBDK de *Nutaq* qui comportent des blocs de *Xilinx System generator (XSG)* dans leur modèle de conception. C'est pourquoi l'installation du logiciel ADP de *Nutaq* établit un lien avec le *System generator* de *Xilinx* préalablement associé à *Simulink*. Grâce à ce lien, le bloc *System generator* peut cibler les produits de *Nutaq*.

Enfin, la bibliothèque MBDK est organisée en deux sections : la première comprend les blocs de conception FPGA colorés en vert et la seconde contient les blocs de couleur jaune, couleur utilisée en général pour la réalisation d'interface utilisateur sur *Simulink* et donc qui ne consomme aucune ressource dans le FPGA. La figure 3.9 montre un exemple de fenêtre *Simulink* incluant des blocs de la bibliothèque MBDK de *Nutaq*.

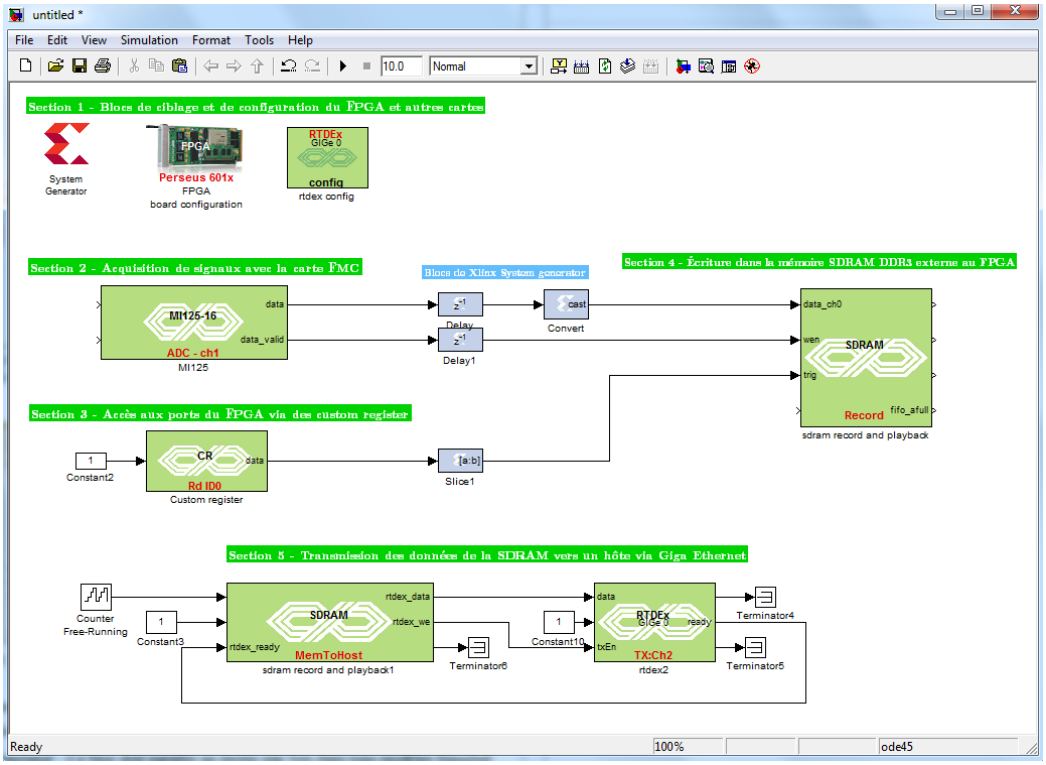


FIGURE 3.9 – Synoptique d'une fenêtre *Simulink* avec l'approche MBDK de *Nutaq*

3.5 Conclusion

L'ensemble du matériel présenté dans ce chapitre, mis à part la *PicoDigitizer* est utilisé pour émuler un environnement de test respectant la problématique du projet. En émission, le matériel utilisé permet de générer des signaux indépendants autour de 10 GHz faisant office de source. À la réception, le montage

présenté est destiné à baisser suffisamment la fréquence des signaux sources avant échantillonnage par la carte FMC interne à la *PicoDigitizer*. Cette dernière a également été discutée, car elle est responsable de l'exécution de l'algorithme MUSIC.

Le projet souscrivant à l'approche MBD, les logiciels y afférents ont donc été initiés. Ces logiciels sont : *Simulink*, la suite ISE de *Xilinx* avec *System generator* et *ADP-MicroTCA*. L'environnement de développement utilisé est *Simulink*. Les deux autres logiciels rajoutent des bibliothèques dans *Simulink* qui serviront à la conception du FPGA. Ainsi, le bénéfice de l'utilisation des MBD est respecté à savoir gain en temps de conception, bonne lisibilité du code, simulation avec le matériel à chaque étape de la conception.

Chapitre 4

Solutions théoriques et aspect général du système adopté

4.1 Introduction

La détermination des angles d'arrivée des signaux reçus par l'antenne réseau via l'implantation de l'algorithme MUSIC sur FPGA est l'apanage d'un processus de module en chaine. Le processus général commence par l'acquisition des signaux en phase et quadrature et le calcul de la matrice de covariance complexe. La matrice de covariance complexe ainsi obtenue est transformée en matrice de covariance réelle avant d'être décomposée en valeurs et vecteurs propres. Sur la base de traitement des valeurs et vecteurs propres, le pseudo-spectre de MUSIC est déterminé mettant en évidence les angles d'arrivée des sources incidentes sur le réseau d'antennes. Ce chapitre présente les solutions théoriques adoptées pour chacune des étapes de la chaine de réalisation du projet.

4.2 Matrice de covariance

Le calcul de la matrice de covariance fait suite à l'acquisition des signaux incidents sur le réseau d'antennes et est le socle permettant la détermination des angles d'arrivée. En accord avec le chapitre sur les notions théoriques, la matrice de covariance est calculée par la relation (2.20). Puisque la matrice de données $\mathbf{X}$ est complexe et composée de signaux provenant d'un réseau d'antennes linéaire uniforme, d'après [15] la forme de la matrice de covariance $\mathbf{R}_{xx}$ est complexe hermitienne et persymétrique. C'est-à-dire que la matrice respecte $\mathbf{R}^{\dagger} = \mathbf{R}$. Grâce à la forme symétrique de la matrice de covariance, seulement la partie supérieure de celle-ci est calculée afin de réduire le nombre de calculs.

4.3 Transformée unitaire

La transformée unitaire (unitary transform) convertit la matrice de covariance complexe persymétrique hermitienne en une matrice réelle symétrique hermitienne. Cette nouvelle matrice obtenue permet

des calculs de décomposition EVD avec nombres réels qui serviront à déduire la EVD de la matrice complexe. Ainsi, la quantité de calculs est considérablement réduite. La transformée unitaire est réalisée par l'équation (4.1)

$$\mathbf{R}'_{xx} = \mathrm{Re}[\mathbf{U}\mathbf{R}_{xx}\mathbf{U}^{\dagger}] \tag{4.1}$$

où Re[.] désigne la partie réelle et **U** représente la matrice unitaire. Cette dernière est définie en fonction de la parité du nombre d'éléments N du réseau et donc de la taille de la matrice de covariance. Pour N paire, elle s'exprime comme suit

$$\mathbf{U} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{I} & \mathbf{J} \\ j\mathbf{J} & -j\mathbf{I} \end{bmatrix} \tag{4.2}$$

avec **J** représentant une matrice antidiagonale unitaire avec des 1 sur la contre-diagonale et 0 ailleurs, et **I** désignant la matrice identité, toutes les deux de taille $\frac{N}{2} \times \frac{N}{2}$.

$$\mathbf{J} = \begin{bmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{bmatrix}. \tag{4.3}$$

Lorsque N est impair, la taille de **J** et **I** est de $\frac{N-1}{2} \times \frac{N-1}{2}$ et **U** prend la forme matricielle suivante :

$$\mathbf{U} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{I} & 0 & \mathbf{J} \\ 0 & \sqrt{2} & 0 \\ j\mathbf{J} & 0 & -j\mathbf{I} \end{bmatrix}. \tag{4.4}$$

D'après [6], les valeurs propres λ_i de la matrice de covariance originale complexe sont sensiblement égales aux valeurs propres issues de la matrice de covariance réelle λ'_i. Par contre, une transformée inverse des vecteurs propres réels $\mathbf{v}'_i$ est nécessaire pour égaler les vecteurs propres complexes originaux $\mathbf{v}_i$.

$$\lambda_i \approx \lambda'_i \tag{4.5}$$

$$\mathbf{v}_i \approx \mathbf{U}^{\dagger}\mathbf{v}'_i \tag{4.6}$$

4.4 Décomposition en valeurs et vecteurs propres (EVD) : Méthode de Jacobi

La plus importante opération à la détermination des angles d'arrivée, la décomposition en valeurs et vecteurs propres d'une matrice de covariance $\mathbf{R}_{xx}$ est une factorisation de la forme :

$$\mathbf{R}_{xx} = \mathbf{V}\mathbf{D}\mathbf{V}^{\dagger} \tag{4.7}$$

où $\mathbf{V}$ est une matrice orthogonale contenant les vecteurs propres, $\mathbf{D}$ une matrice diagonale représentant l'ensemble des valeurs propres réelles, avec $\mathbf{R}_{xx}$ hermitienne. Le traitement de la EVD d'une matrice par le biais d'un FPGA n'est pas très commun et les techniques proposées dans la littérature se résument en deux classes. La première transforme la structure initiale de la matrice et devient itérative par la suite [2] (Muller, QR, QZ, etc.). La seconde est uniquement itérative (Jacobi, Givens). Parmi ces dernières méthodes, celle basée sur le modèle proposé par Jacobi a une structure modulaire facilitant son implémentation dans les structures matérielles à traitement parallèle tel que le FPGA. C'est cette méthode qui a été utilisée dans ce projet.

L'algorithme de Jacobi est une méthode itérative qui génère les matrices $\mathbf{V}$ et $\mathbf{D}$ en effectuant une séquence de rotations dans le plan orthogonal des deux côtés de la matrice de covariance, de sorte que chaque nouvelle matrice $\mathbf{R}_{xx}^{(h+1)}$ soit plus diagonale que la précédente, en prenant

$$\mathbf{R}_{xx}^{(h+1)} = \mathbf{S}(\theta^{(h)})^{\top}\mathbf{R}_{xx}^{(h)}\mathbf{S}(\theta^{(h)}) \tag{4.8}$$

avec $\mathbf{S}$, la matrice de rotation proposée par Wallace Givens et communément appelée matrice de Givens donnée par (4.9). Cette matrice est égale à la matrice identité à l'exception des éléments $s_{i,i} = s_{j,j} = \cos(\theta^{(h)})$ et $s_{i,j} = -s_{j,i} = \sin(\theta^{(h)})$, où i correspond à la ligne, j à la colonne d'élément hors diagonale de la matrice $\mathbf{R}_{xx}$. Les angles $\theta^{(h)}$ sont choisis dans le but de mettre à zéro les éléments hors diagonale de la matrice de $\mathbf{R}_{xx}$.

$$\mathbf{S}(\theta^{(h)}) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & \cos(\theta^{(h)}) & \cdots & \sin(\theta^{(h)}) & & 0 \\ 0 & & -\sin(\theta^{(h)}) & \cdots & \cos(\theta^{(h)}) & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}. \tag{4.9}$$

Après H itérations, la matrice d'entrée $\mathbf{R}_{xx}^{(H)}$ est transformée en matrice diagonale $\mathbf{D}$ contenant les

valeurs propres et la matrice rotation $\mathbf{S}(\theta^{(H)})$ contenant sur chaque colonne les vecteurs propres associés.

Pour mieux illustrer le fonctionnement de la méthode de Jacobi, soit une matrice symétrique hermitienne $\mathbf{R} \in \Re^{2\times 2}$ multipliée à gauche et à droite par la matrice de rotation $\mathbf{S} \in \Re^{2\times 2}$ donnée par (4.10).

$$\mathbf{R}^{(1)} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}^{(0)^\top} \begin{bmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{bmatrix}^{(0)} \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}^{(0)} \tag{4.10}$$

Pour que $\mathbf{R}^{(1)}$ soit diagonale, on doit avoir $r_{1,2}^{(1)} = r_{2,1}^{(1)} = 0$.

$$r_{1,2}^{(1)} = 2\, r_{1,2}^{(0)} \cos(2\theta^{(0)}) + (r_{1,1}^{(0)} - r_{2,2}^{(0)}) \sin(2\,\theta^{(0)}) = 0 \tag{4.11}$$

$$\begin{aligned} \tan(2\theta^{(0)}) &= \frac{2\, r_{1,2}^{(0)}}{(r_{2,2}^{(0)} - r_{1,1}^{(0)})} \\ \theta^{(0)} &= \frac{1}{2} \tan^{-1} \left(\frac{2\, r_{1,2}^{(0)}}{r_{2,2}^{(0)} - r_{1,1}^{(0)}} \right) \end{aligned} \tag{4.12}$$

Après une itération de double rotation avec l'angle $\theta^{(0)}$, on obtient la matrice diagonale $\mathbf{R}^{(1)}$ équivalente aux valeurs propres $r_{1,1}^{(1)} = \lambda_1$ et $r_{2,2}^{(1)} = \lambda_2$. Les colonnes $\mathbf{s}_1(\theta^{(0)})$ et $\mathbf{s}_2(\theta^{(0)})$ de la matrice de rotation $\mathbf{S}$ correspondent respectivement aux vecteurs propres $\mathbf{v}_1$ et $\mathbf{v}_2$ associés à λ_1 et λ_2.

4.4.1 Calcul des valeurs propres

Le problème de décomposition en valeurs propres d'une matrice symétrique hermitienne de taille $N \times N$ devient de plus en plus important au fur et à mesure que le nombre d'éléments N est grand. Il requiert alors plusieurs itérations et le nombre de calculs devient considérable. La méthode retenue dans ce projet est basée sur l'algorithme de Jacobi et est proposée par Brent [13]. Elle permet une décomposition du problème en sous-problèmes en vue de réduire le volume de traitement. En effet, la matrice originale complète (4.13) $\mathbf{R} \in \Re^{N\times N}$ avec $N > 2$ est subdivisée en sous-matrices de taille $\mathbf{R}_{i,j} \in \Re^{2\times 2}$ (4.14). Toujours selon [13], le nombre d'itérations nécessaires à la diagonalisation de $\mathbf{R}$ est environ $N\log(N)$.

$$\mathbf{R}^{(h)} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,N-1} & r_{1,N} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,N-1} & r_{2,N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{N,1} & a_{N,2} & \cdots & r_{N,N-1} & r_{N,N} \end{pmatrix} \tag{4.13}$$

$$\mathbf{R}^{(h)} = \begin{pmatrix} \begin{pmatrix} r_{1,1}^{(h)} & r_{1,2}^{(h)} \\ r_{2,1}^{(h)} & r_{2,2}^{(h)} \end{pmatrix} & \begin{pmatrix} r_{1,3}^{(h)} & r_{1,4}^{(h)} \\ r_{2,3}^{(h)} & r_{2,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} r_{1,N-1}^{(h)} & r_{1,N}^{(h)} \\ r_{2,N-1}^{(h)} & r_{2,N}^{(h)} \end{pmatrix} \\ \begin{pmatrix} r_{3,1}^{(h)} & r_{3,2}^{(h)} \\ r_{4,1}^{(h)} & r_{4,2}^{(h)} \end{pmatrix} & \begin{pmatrix} r_{3,3}^{(h)} & r_{3,4}^{(h)} \\ r_{4,3}^{(h)} & r_{4,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} r_{3,N-1}^{(h)} & r_{3,N}^{(h)} \\ r_{4,N-1}^{(h)} & r_{4,N}^{(h)} \end{pmatrix} \\ \vdots & \vdots & \ddots & \vdots \\ \begin{pmatrix} r_{N-1,1}^{(h)} & r_{N-1,2}^{(h)} \\ r_{N,1}^{(h)} & r_{N,2}^{(h)} \end{pmatrix} & \begin{pmatrix} r_{N-1,3}^{(h)} & r_{N-1,4}^{(h)} \\ r_{N,3}^{(h)} & r_{N,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} r_{N-1,N-1}^{(h)} & r_{N-1,N}^{(h)} \\ r_{N,N-1}^{(h)} & r_{N,N}^{(h)} \end{pmatrix} \end{pmatrix} \tag{4.14}$$

$$\mathbf{R}_{i,j}^{(h)} = \begin{bmatrix} r_{2i-1,2j}^{(h)} & r_{2i-1,2j}^{(h)} \\ r_{2i,2j-1}^{(h)} & r_{2i,2j}^{(h)} \end{bmatrix} \quad \text{avec } i,j = 1,\cdots,\frac{N}{2} \tag{4.15}$$

Les sous-matrices $\mathbf{R}_{i,j}^{(h)}$ sont classées en deux groupes : les sous-matrices diagonales $i = j$ qui sont symétriques et les sous-matrices non-diagonales $i \neq j$ qui elles sont asymétriques. Cette distinction des sous-matrices en deux classes est importante, car uniquement les sous-matrices diagonales verront leurs éléments diagonaux $r_{2i-1,2i}^{(h)} = r_{2i,2i-1}^{(h)}$ mis à zéro à chaque itération.

Chaque sous-matrice de la matrice complète $\mathbf{R}$ effectue une double rotation, comme le montre l'équation suivante

$$\mathbf{R}^{(h+1)} = \mathbf{S}(\theta_{i,i}^{(h)})^{\top} \mathbf{R}_{i,j}^{(h)} \mathbf{S}(\theta_{j,j}^{(h)}) \quad i,j = 1,\cdots,\frac{N}{2} \quad \text{et} \quad h = 1,\cdots,N\log N. \tag{4.16}$$

On remarque une petite différence entre l'équation (4.16) et l'équation (4.10). Contrairement à l'équation (4.10) où la matrice de rotation à gauche équivaut à celle de droite, l'équation (4.16) peut avoir une matrice de rotation à gauche différente de celle à droite puisque $\theta_{i,i}$ peut être différent de $\theta_{j,j}$. La raison de cette différence est la prise en compte des deux types de sous-matrices : symétrique (diagonale) où $\mathbf{S}(\theta)$ ne varie pas, asymétrique (non-diagonale) où $\mathbf{S}(\theta)$ varie. Concrètement, la matrice de rotation $\mathbf{S}(\theta)$ varie en fonction du type de sous-matrice car $\theta_{i,i} = \theta_{j,j}$ si $i = j$ et $\theta_{i,i} \neq \theta_{j,j}$ si $i \neq j$.

Puisque les sous-matrices non-diagonales $\mathbf{R}_{i,j}$ avec $i \neq j$ ne sont pas symétriques, leurs éléments non-diagonaux $r_{2i-1,2j}$ et $r_{2i,2j-1}$ ne peuvent être mis à zéro suite à une double rotation quel que soit l'angle θ, contrairement aux sous-matrices diagonales. C'est pourquoi, seulement les angles $\theta_{i,i}$ et $\theta_{j,j}$ liés aux sous-matrices diagonales sont calculés. La rotation des sous-matrices non diagonales est effectuée avec les angles de leur sous-matrice diagonale associée. L'association ici signifie être sur la même ligne i et la même colonne j que la sous-matrice diagonale comme sur la figure 4.1.

Une fois la double rotation effectuée pour chaque sous-matrice avec les angles correspondants, les résultats partiels obtenus sont réordonnés adéquatement à l'intérieur de chaque sous-matrice et transférés vers les sous-matrices voisines dans le but de positionner de nouveaux éléments diagonaux en

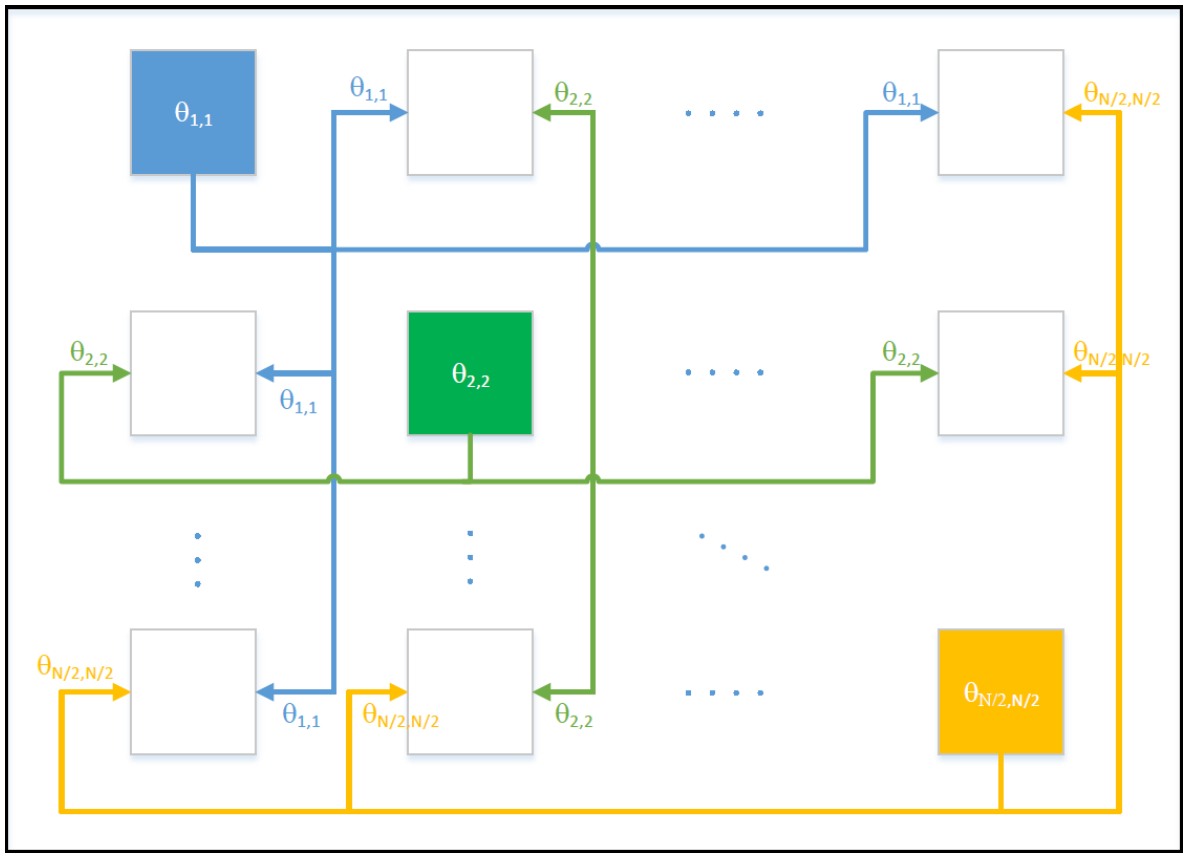


FIGURE 4.1 – Propagation des angles des sous-matrices diagonales aux non-diagonales adjacantes

vue de les mettre à zéro lors de la prochaine itération. La figure 4.2 récapitule le fonctionnement de la méthode de Jacobi couplée à la méthode de Brent pour le calcul des valeurs propres.

4.4.2 Décomposition en vecteurs propres

Comme précédemment indiqué, la matrice $\mathbf{V} \in \Re^{(N \times N)}$ contenant les vecteurs propres associés aux valeurs propres de $\mathbf{R} \in \Re^{(N \times N)}$ est issue d'une accumulation à chaque itération h des différentes valeurs de la matrice de rotation. C'est pourquoi la matrice de départ utilisée pour le calcul des vecteurs propres est la matrice identité $\mathbf{V}^{(h=0)} = \mathbf{I} \in \Re^{(N \times N)}$ de même taille que la matrice de covariance utilisée à l'étape du calcul des valeurs propres. À l'instar du calcul des valeurs propres la matrice de départ sera décomposée en sous-matrices de 2×2 éléments $\mathbf{V}_{i,j}^{(h)} \in \Re^{(2 \times 2)}$. Aussi, il n'est pas nécessaire de faire une distinction entre les sous-matrices diagonales et non-diagonales, car seule la matrice identité de départ utilisée est symétrique.

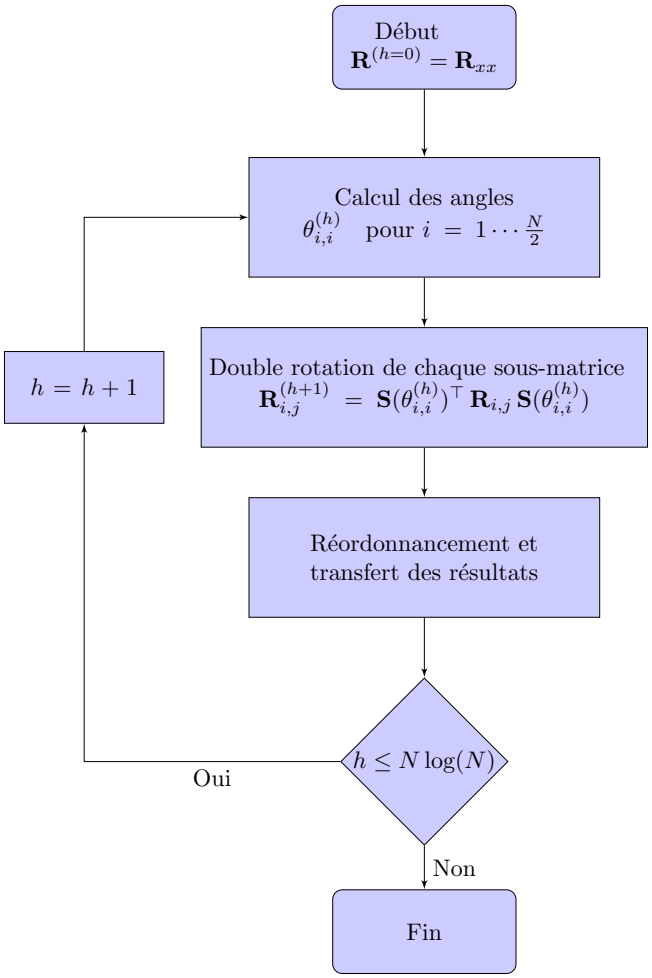


FIGURE 4.2 – Diagramme de flux pour le calcul des valeurs propres avec la méthode de Jacobi

$$\mathbf{V}^{(h)} = \begin{pmatrix} \begin{pmatrix} v_{1,1}^{(h)} & v_{1,2}^{(h)} \\ v_{2,1}^{(h)} & v_{2,2}^{(h)} \end{pmatrix} & \begin{pmatrix} v_{1,3}^{(h)} & v_{1,4}^{(h)} \\ v_{2,3}^{(h)} & v_{2,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} v_{1,N-1}^{(h)} & v_{1,N}^{(h)} \\ v_{2,N-1}^{(h)} & v_{2,N}^{(h)} \end{pmatrix} \\ \begin{pmatrix} v_{3,1}^{(h)} & v_{3,2}^{(h)} \\ v_{4,1}^{(h)} & v_{4,2}^{(h)} \end{pmatrix} & \begin{pmatrix} v_{3,3}^{(h)} & v_{3,4}^{(h)} \\ v_{4,3}^{(h)} & v_{4,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} v_{3,N-1}^{(h)} & v_{3,N}^{(h)} \\ v_{4,N-1}^{(h)} & v_{4,N}^{(h)} \end{pmatrix} \\ \vdots & \vdots & \ddots & \vdots \\ \begin{pmatrix} v_{N-1,1}^{(h)} & v_{N-1,2}^{(h)} \\ v_{N,1}^{(h)} & v_{N,2}^{(h)} \end{pmatrix} & \begin{pmatrix} v_{N-1,3}^{(h)} & v_{N-1,4}^{(h)} \\ v_{N,3}^{(h)} & v_{N,4}^{(h)} \end{pmatrix} & \cdots & \begin{pmatrix} v_{N-1,N-1}^{(h)} & v_{N-1,N}^{(h)} \\ v_{N,N-1}^{(h)} & v_{N,N}^{(h)} \end{pmatrix} \end{pmatrix} \tag{4.17}$$

$$\mathbf{V}_{i,j}^{(h)} = \begin{bmatrix} v_{2i-1,2j}^{(h)} & v_{2i-1,2j}^{(h)} \\ v_{2i,2j-1}^{(h)} & v_{2i,2j}^{(h)} \end{bmatrix} \quad \text{avec } i,j = 1,\cdots,\frac{N}{2} \tag{4.18}$$

$$\mathbf{V}_{i,j}^{(h+1)} = \mathbf{V}_{i,j}^{(h)} \mathbf{S}(\theta_{j,j}^{(h)}) \quad \text{avec } i,j = 1,\cdots,\frac{N}{2} \quad \text{et } h = 1,\cdots,N\log N \tag{4.19}$$

Les angles $\theta_{j,j}$ nécessaires pour le calcul des vecteurs propres sont générés à partir des sous-matrices diagonales dans l'étape des valeurs propres. Aussi, chaque colonne de sous-matrice de vecteurs propres partage le même angle généré par la sous-matrice diagonale de même colonne dans l'étape des valeurs propres comme à la figure 4.3. En d'autres termes, l'angle $\theta_{j,j}^{(h)}$ calculé à partir de la sous-matrice diagonale $\mathbf{R}_{j,j}^{(h)}$ dans l'étape des valeurs propres, est utilisée pour le calcul en (4.19) des sous-matrices vecteurs de même colonne soit j ($[v_{1,j}^{(h)} v_{2,j}^{(h)} \dots v_{\frac{N}{2},j}^{(h)}]$; $\quad j = 1,2 \dots \frac{N}{2}$).

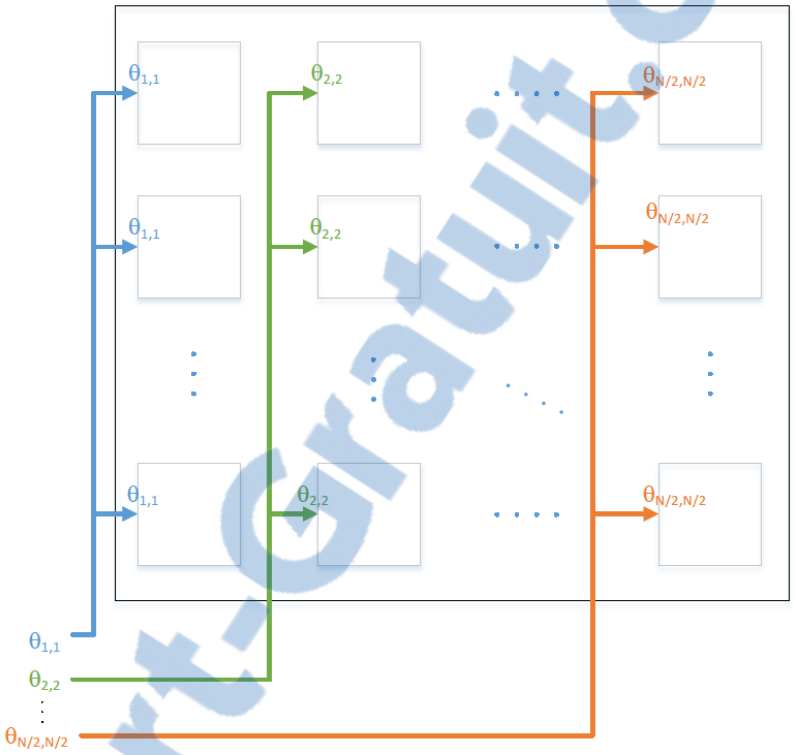


FIGURE 4.3 – Propagation des angles aux sous-matrices vecteurs

Une fois les angles de chaque sous-matrice disponibles et l'équation (4.19) réalisée pour chaque sous-matrice vecteur, le même processus indiqué précédemment dans l'étape des valeurs propres, à savoir le réordonnancement interne des sous-matrices et le transfert des leurs éléments, est exécuté. Le chapitre suivant donne des précisions sur le déroulement de cet ordonnancement en question des différentes sous-matrices. Un résumé de ce qui est expliqué dans cette section est illustré par le diagramme de flux à la figure 4.4.

4.4.3 Réordonnancement des éléments des matrices partielles de vecteurs et valeurs propres

La relocalisation des éléments des matrices $\mathbf{V}^{(h+1)}$ et $\mathbf{R}_{xx}^{(h+1)}$ de vecteurs et de valeurs propres est une opération importante dans la méthode de Jacobi. En effet, elle permet de repositionner de nouveaux éléments de la matrice partielle des valeurs propres sur sa diagonale en vue de les mettre à zéro à

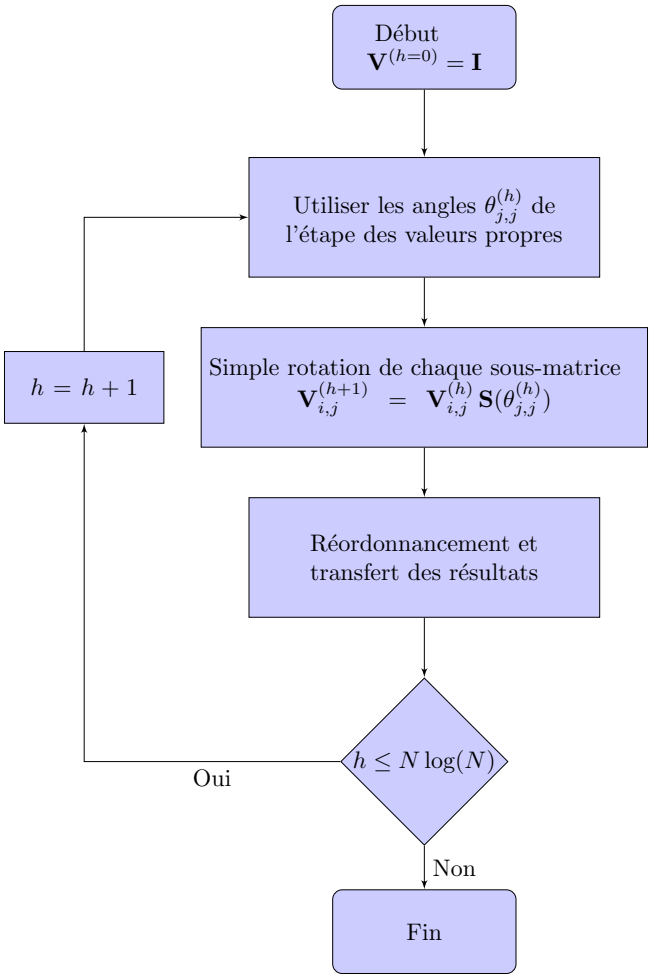


FIGURE 4.4 – Diagramme de flux pour le calcul des vecteurs propres avec la méthode de Jacobi

l'itération suivante. Le réordonnancement est effectué de manière identique pour les deux matrices de vecteurs et valeurs propres.

Les auteurs de [13] effectuent la relocalisation des données partielles dans chaque sous-matrice et leur transfert vers les voisines en utilisant une architecture systolique. Il s'agit d'une architecture dans laquelle le terme sous-matrice est remplacé par processeur. Cela afin d'idéaliser les sous-matrices de sorte qu'elles puissent diffuser leurs éléments (les paramètres de rotations). Ainsi, "Processeur Diagonal (PD)" pour les sous-matrices $\mathbf{R}_{i,i}^{(h)}$ et "Processeur Non-Diagonal (PND)" pour les sous-matrices $\mathbf{R}_{i,j}^{(h)}$ dans le cadre du calcul des valeurs propres ; dans le cadre des vecteurs propres, le seul type de sous-matrice utilisé est remplacé par "Processeur Vectoriel (PV)" tout simplement.

Le réordonnancement se résume en 2 types de déplacements : le déplacement d'éléments à l'intérieur de chaque sous-matrice (déplacement interne) ; l'échange d'éléments entre les sous-matrices adjacentes ou voisines (déplacement externe). L'ordre d'exécution de ces déplacements est important. Le déplacement externe est effectué avant le déplacement interne. Pour expliquer en quoi consiste ces déplacements,

les figures 4.5 et 4.6 aident en prenant comme exemple une matrice de taille 8×8, taille similaire à la matrice de covariance utilisée dans ce projet.

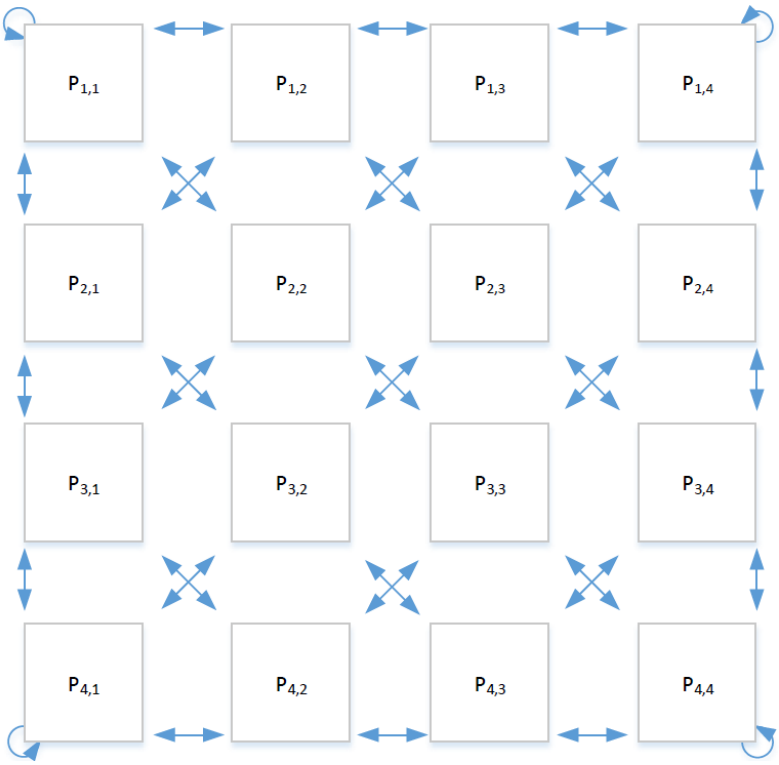


FIGURE 4.5 – Déplacement externe : Échanges entre les processeurs

À partir de la figure 4.5 illustrant les déplacements possibles entre les processeurs, on dénombre quatre différents types de propagation :

— Les quatre processeurs situés aux quatre coins : chacun de ces processeurs a un de ses éléments qui n'est pas transmis aux voisins.

— Les processeurs situés sur la première et la dernière colonnes : les deux éléments de gauche de chaque processeur ont un déplacement vertical et les deux de droite ont un déplacement oblique.

— Les processeurs situés sur la première et la dernière lignes : les deux éléments supérieurs de chaque processeur ont un déplacement horizontal et les deux inférieurs ont déplacement oblique.

— Les autres processeurs : le déplacement des éléments de chaque processeur est oblique.

Le déplacement interne des éléments dans chaque processeur à l'instar du déplacement externe peut être décomposé en quatre différents types de propagation :

— Le processeur situé dans le coin supérieur gauche $\mathbf{P}_{1,1}$: aucun échange entre les éléments.

— Les processeurs situés sur la première colonne : les éléments échangent de position verticalement.

— Les processeurs situés sur la première ligne : les éléments échangent de position horizontalement.

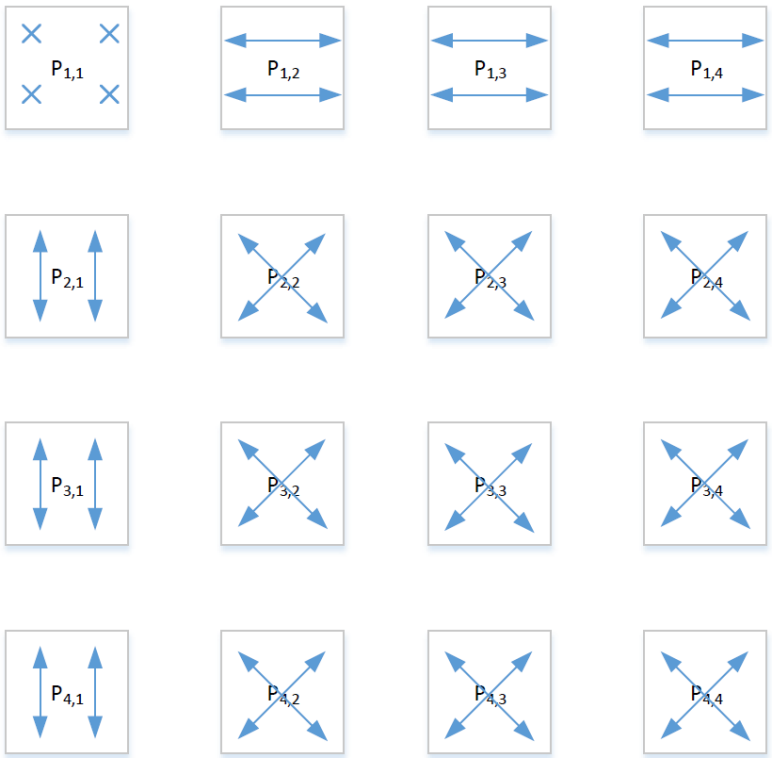


FIGURE 4.6 – Déplacement à l'intérieur de chaque processeur

— Les autres processeurs : les éléments échangent de position de manière croisée.

En fonction du type de déplacement, ces différentes catégories de propagation représentent les conditions implémentées par le programme FPGA afin d'effectuer les déplacements. Par exemple, si chaque processeur a la forme suivante :

$$\mathbf{P} = \begin{bmatrix} \alpha_{i,j} & \beta_{i,j} \\ \gamma_{i,j} & \delta_{i,j} \end{bmatrix} \tag{4.20}$$

les conditions pour un déplacement externe de l'élément $\alpha_{i,j}$ du processeur $\mathbf{P}_{i,j}$ peuvent être comme ceci :

$$\text{la sortie de } \alpha_{i,j} \text{ est reliée} \begin{cases} \text{à l'entrée de} & \delta_{i-1,j-1} & \text{si } i > 1 \text{ et } j > 1 \\ \text{à l'entrée de} & \beta_{i,j-1} & \text{si } i = 1 \text{ et } j > 1 \\ \text{à l'entrée de} & \gamma_{i-1,j} & \text{si } i > 1 \text{ et } j = 1 \\ \text{à l'entrée de} & \alpha_{i,j} & \text{si } i = 1 \text{ et } j = 1 \end{cases}$$

L'algorithme de déplacement interne est défini comme suit :

$$\begin{array}{lll}
\text{si } i=1 \text{ et } j=1 & \text{alors} & \begin{bmatrix} \text{sortie } \alpha \leftarrow \alpha & \text{sortie } \beta \leftarrow \beta \\ \text{sortie } \gamma \leftarrow \gamma & \text{sortie } \delta \leftarrow \delta \end{bmatrix} \\
\text{sinon si } i=1 & \text{alors} & \begin{bmatrix} \text{sortie } \alpha \leftarrow \beta & \text{sortie } \beta \leftarrow \alpha \\ \text{sortie } \gamma \leftarrow \delta & \text{sortie } \delta \leftarrow \gamma \end{bmatrix} \\
\text{sinon si } j=1 & \text{alors} & \begin{bmatrix} \text{sortie } \alpha \leftarrow \gamma & \text{sortie } \beta \leftarrow \delta \\ \text{sortie } \gamma \leftarrow \alpha & \text{sortie } \delta \leftarrow \beta \end{bmatrix} \\
\text{sinon} & & \begin{bmatrix} \text{sortie } \alpha \leftarrow \delta & \text{sortie } \beta \leftarrow \gamma \\ \text{sortie } \gamma \leftarrow \beta & \text{sortie } \delta \leftarrow \alpha \end{bmatrix}
\end{array}$$

.

4.5 Tri et réduction des valeurs et vecteurs propres

L'algorithme MUSIC se base de préférence sur la projection dans le sous-espace bruit engendré par les vecteurs propres bruit de la matrice de covariance pour un pseudo spectre plus fin. On rappelle que la matrice des vecteurs propres est constituée des vecteurs propres sources associés aux valeurs propres significatives et des vecteurs propres bruit associés aux valeurs propres nulles de $\mathbf{R}_{xx-}$. Les vecteurs propres bruit étant difficiles à obtenir [1], ils sont retrouvés à partir des vecteurs sources grâce à l'orthogonalité entre les deux sous-espaces.

Une fois la décomposition en valeurs et vecteurs propres terminée les vecteurs propres de même que les valeurs propres obtenues ne sont pas nécessairement ordonnés. Il est important de les classer en ordre décroissant (ou croissant) afin d'identifier les M premières (dernières) valeurs propres associées aux M sources incidentes sur le réseau d'antennes et extraire leurs vecteurs sources propres à traiter. Le nombre de sources M est déterminé au moyen de l'équation (4.21) plutôt que sur le critère de Akaike, pour l'instant.

$$\frac{\sum_{k=1}^{M}(\lambda_k)}{\sum_{k=1}^{N}(\lambda_k)} \leq 0,95 \tag{4.21}$$

La méthode du tri à bulles a été utilisée pour la classification en ordre décroissant des valeurs propres et leurs vecteurs propres associés. C'est une méthode itérative optimale dans le traitement par FPGA. Pour le cas du projet, le nombre d'éléments ou le nombre de valeurs propres vaut $N = 8$. Après avoir

1. Cela est dû aux valeurs propres bruit qui sont identiques ou presque.

Acc0	Acc1	Acc2	Acc3	Acc4	Acc5	Acc6	Acc7
λ_1	λ_5	λ_3	λ_7	λ_0	λ_6	λ_4	λ_2

TABLE 4.1 – Exemple illustratif de la position et du contenu d'une mémoire avec les valeurs propres obtenues par la méthode de Jacobi

appliqué la technique de Jacobi, les valeurs propres obtenues sont enregistrées dans une mémoire selon un ordre quelconque, comme celui dans le tableau 4.1. Ainsi, λ_7 représente la valeur propre la plus élevée et λ_0 la valeur la plus faible.

4.6 Pseudo spectre

La dernière opération de l'algorithme MUSIC est le calcul du pseudo-spectre. Ce dernier permet la visualisation graphique des angles d'arrivée. Il fait suite à la classification en ordre décroissant des valeurs et vecteurs propres. Le pseudo-spectre s'obtient par la projection du vecteur de balayage selon θ dans le sous-espace bruit. Cette projection est en réalité le carré de la distance euclidienne ($d_s^2(\theta)$) entre le $\mathbf{a}(\theta)$ et le sous-espace bruit $\mathbf{V}_n$ comme exprimé par (2.22).

$$d_s^2(\theta) = \frac{1}{N}\mathbf{a}_\theta^\dagger \underbrace{\mathbf{V}_n\mathbf{V}_n^\dagger}_{\mathbf{P}_n}\mathbf{a}_\theta \tag{4.22}$$

En pratique les vecteurs propres bruit étant difficiles à déterminer, $\mathbf{P}_n$ est exprimé à partir des vecteurs propres sources selon (4.23). Le vecteur de balayage est fonction de l'angle θ et est exprimé de la même manière qu'un vecteur directionnel de l'équation (2.7). Les angles du vecteur de balayage sont choisis pour couvrir une plage d'angles qui inclue tous les angles d'arrivée potentiels. Parmi ces angles, ceux qui provoquent l'annulation (ou la minimisation dans le cas avec bruit) de $d_s^2(\theta)$ sont les angles d'arrivée recherchés. En effet, lorsque distance euclidienne est nulle (ou minimale avec bruit), le vecteur de balayage se situe dans le sous-espace source et coïncide avec un vecteur directionnel.

$$\mathbf{P}_n = \mathbf{I} - \mathbf{V}_s\mathbf{V}_s^\dagger \tag{4.23}$$

4.7 Calibrations des données

La théorie en traitement d'antenne affirme que les signaux reçus par les antennes d'un réseau uniforme linéaire soient de même amplitude et aient le même déphasage. En pratique, on observe une tout autre réalité, comme le montrent les figures 4.8 et 4.9 représentant respectivement les signaux reçus en phase (I) en quadrature (Q). En effet, ces figures illustrent des signaux déphasés entre eux et d'amplitudes différentes.

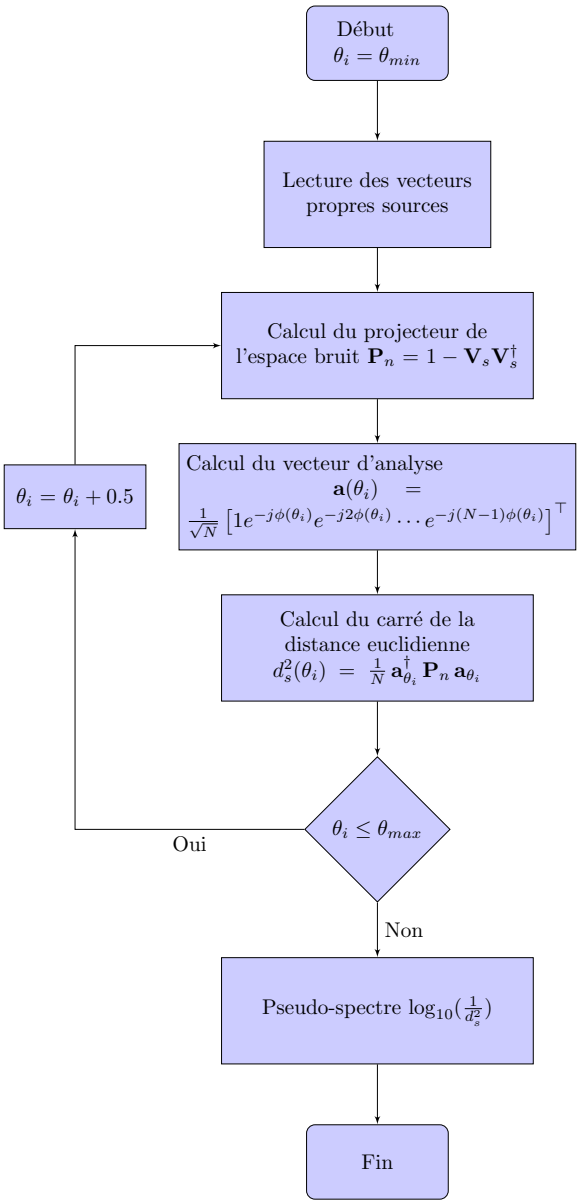


FIGURE 4.7 – Diagramme de flux du processus de calcul du pseudo-spectre

De plus, la même théorie mentionne un déphasage de 90° entre les signaux I et Q. La figure de Lissajou 4.10 des signaux I et Q reçus par les 8 antennes utilisées dans le projet, prouve que les signaux obtenus ne sont pas déphasés de 90°. Ainsi, ils ne respectent pas les recommandations théoriques.

Les résultats des tests effectués via *Matlab* de l'algorithme MUSIC avec ces signaux reçus mènent à la conclusion que les signaux sont inutilisables pour obtenir une haute-résolution. La figure 4.11 est la preuve de cette conclusion.

Tous ces constats sont le fruit de plusieurs causes :

— Déséquilibre de gain causé par le fait que les amplificateurs utilisés dans la chaine de réception

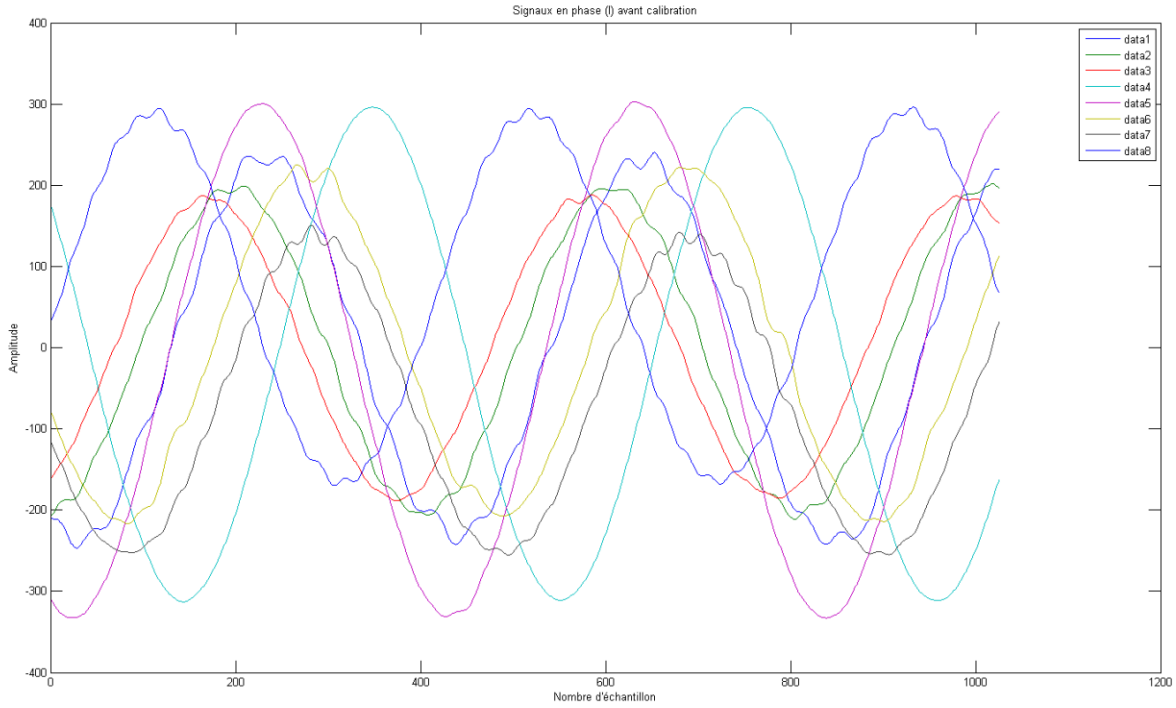


FIGURE 4.8 – Signaux en phase non calibrés d'une source placée à 90 degrés

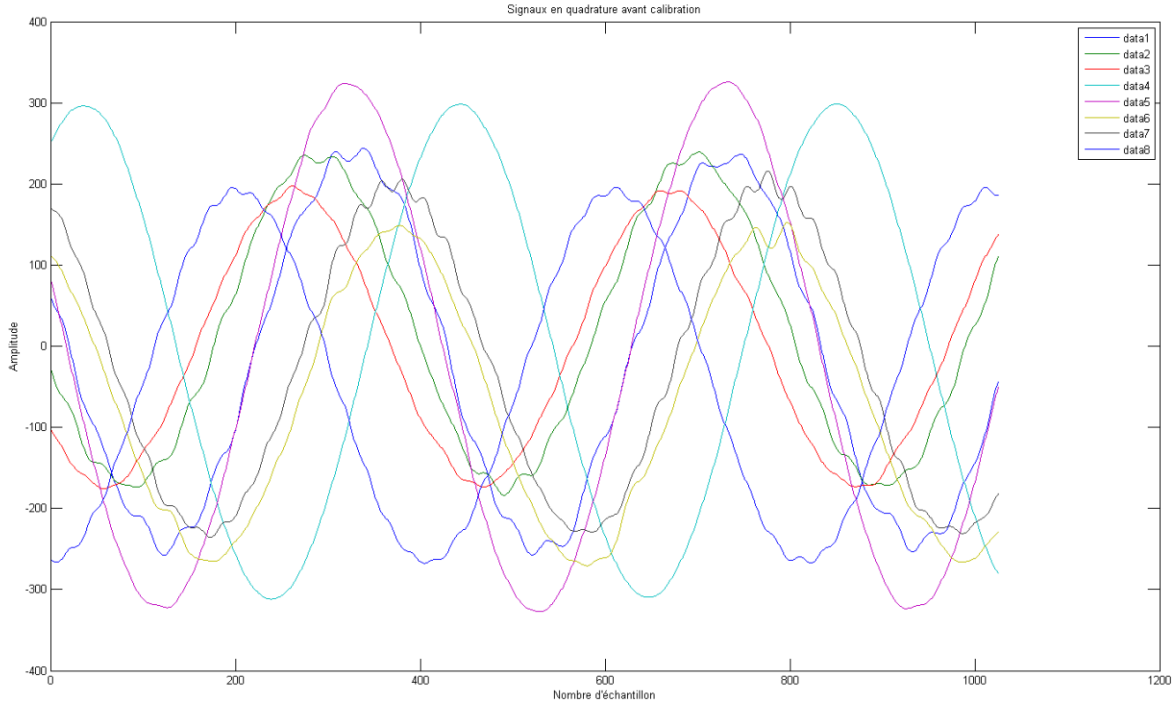


FIGURE 4.9 – Signaux en quadrature non calibrés d'une source placée à 90 degrés

ne sont pas identiques.

— Différence de phase causée par l'imperfection de fabrication des coupleurs hybrides 90°. Le

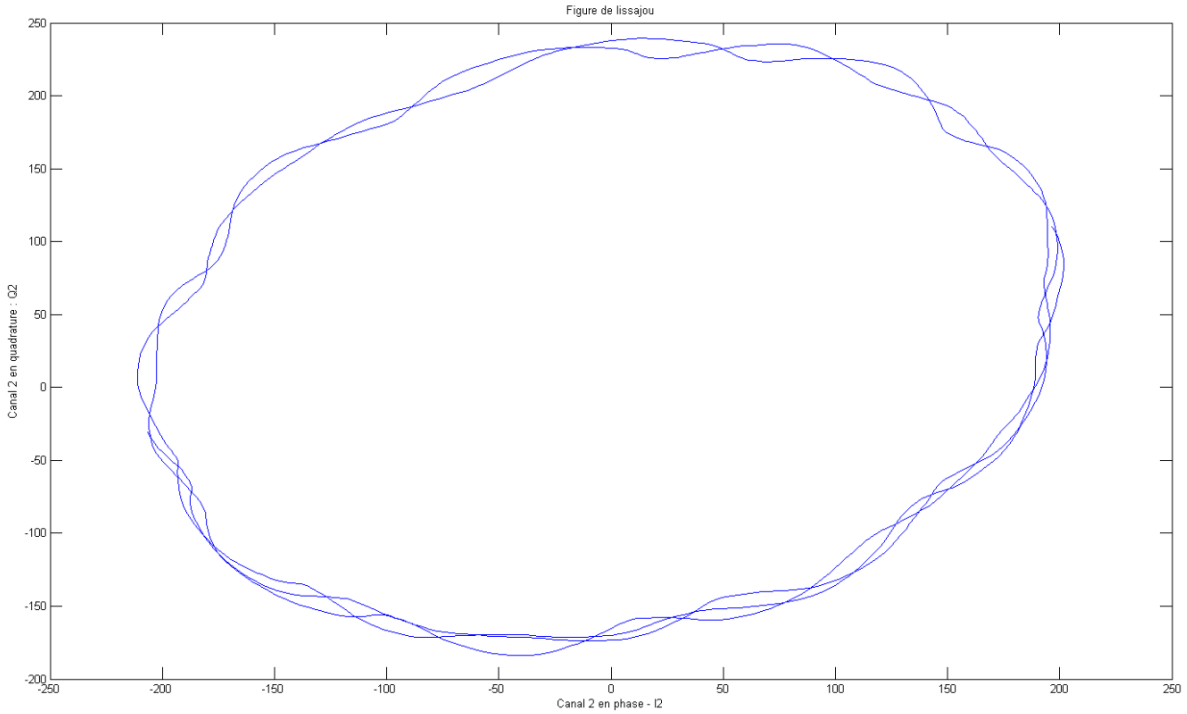


FIGURE 4.10 – Figure de Lissajou des signaux non calibrés en phase et en quadrature du canal 2

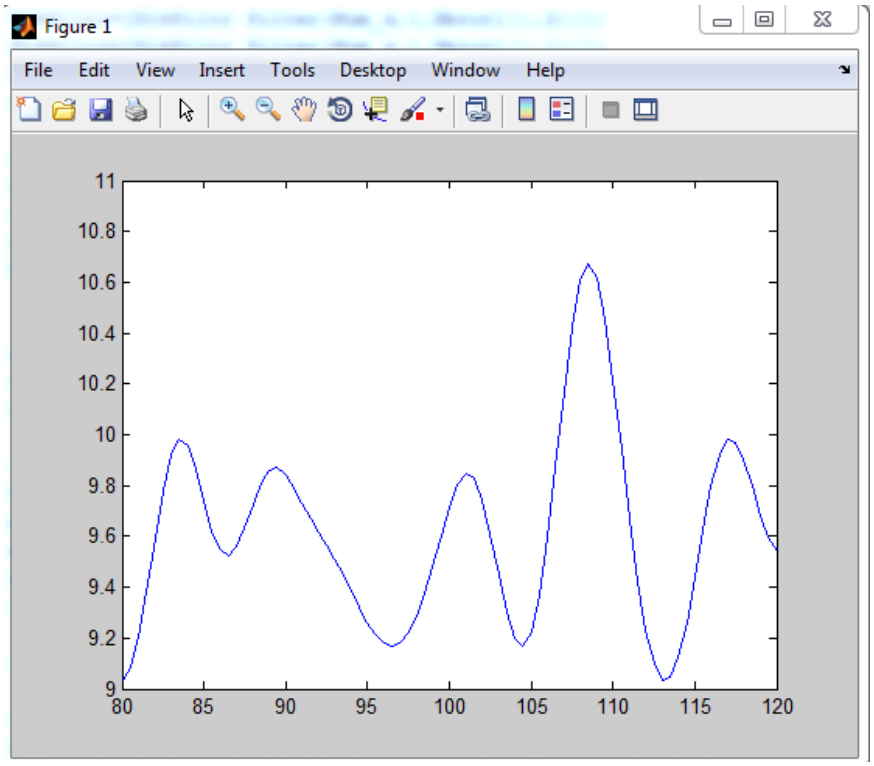


FIGURE 4.11 – Pseudo-spectre des signaux non calibrés obtenu par application de MUSIC

manufacturier indique une incertitude de phase de 3° possible.

— Couplage mutuel entre les antennes du réseau.

— Différence entre les chaines de réception de chaque élément du réseau.

Les conséquences sont immédiates avec un effet d'avalanche générant une corruption de la structure de la matrice de covariance : les vecteurs de balayage même théoriques ne sont plus inclus dans le sous-espace signal procurant une performance médiocre de l'algorithme MUSIC.

Fixer ces problèmes passe inévitablement par la calibration des données reçues. Les techniques de calibration disponibles dans la littérature peuvent être regroupées en trois catégories. La première vise à contourner le problème en reconstituant une nouvelle matrice de covariance ayant les mêmes caractéristiques que celle recherchée. La seconde est la calibration en ligne ou auto calibration sensée en théorie ajuster les données sans connaissance préalable de la position de sources de référence ; dans la pratique, elle démontre qu'elle n'est pas réellement autosuffisante. Enfin, la troisième catégorie est appelée calibration supervisée ; elle nécessite la connaissance de la position de sources de référence.

Dans le projet, nous avons testé deux techniques de calibration appartenant respectivement aux catégories 2 et 3 : calibration par matrice **G** et calibration par réseau de neurones plus précisément le réseau ADALINE. Les résultats obtenus par ces méthodes ont été concluants. La technique ADALINE [5] possède une architecture naturellement parallèle et est très peu couteuse en temps de calcul, ce qui est intéressant pour le traitement en temps réel et son implémentation dans un FPGA. L'inconvénient majeur de cette technique, comme toutes les techniques appartenant à la catégorie 3 décrite précédemment, est qu'elle nécessite un grand nombre de prises de mesures pour son apprentissage. Ceci est fastidieux lorsque ces mesures sont effectuées de façon manuelle. D'un autre côté, la calibration par matrice **G** donne des résultats acceptables, mais de moins bonne qualité que ceux obtenus avec ADALINE. Son avantage par rapport à la technique basée sur le réseau de neurones est qu'elle nécessite uniquement une seule prise de mesures au préalable pour calibrer les données.

4.7.1 Calibration par Matrice G

La calibration par matrice **G** est une technique basée sur les moindres carrées. Elle nécessite la connaissance au préalable de la position d'un signal source de référence pour générer la matrice **G**. Cette dernière est ensuite utilisée pour calibrer les signaux bruts mesurés ou la matrice de covariance de ces signaux. Ce type de calibration est dit d'auto calibration, car il peut s'adapter en modifiant la matrice **G** à un nouvel environnement même après implémentation dans un FPGA.

Le calcul de la matrice **G** part du principe de la minimisation de l'erreur quadratique entre les vecteurs de balayage expérimentaux et leurs équivalents théoriques comme le montre l'équation (4.24) :

$$\hat{\mathbf{G}} = \min_{\mathbf{G}} \|\mathbf{A} - \mathbf{G}\hat{\mathbf{A}}_e\| \tag{4.24}$$

où la matrice $\mathbf{A}$ représente les vecteurs de balayage théoriques. Puisqu'il n'est pas aisé dans la pratique d'extraire les vecteurs de balayage expérimentaux $\mathbf{A}_e$, il est recommandé de les substituer par les vecteurs propres associés aux valeurs propres significatives issues de la décomposition EVD. Ainsi, ces vecteurs propres principaux sont une estimation de $\mathbf{A}_e$ qu'on note ici $\hat{\mathbf{A}}_e$, car ils génèrent le même sous-espace. De même, la matrice $\mathbf{G}$ est aussi une matrice estimée qu'on note $\hat{\mathbf{G}}$.

$$\hat{\mathbf{G}} = \frac{\mathbf{A}\hat{\mathbf{A}}_e^{\dagger}}{\hat{\mathbf{A}}_e\hat{\mathbf{A}}_e^{\dagger}} \tag{4.25}$$

D'après [10], en développant et simplifiant l'équation (4.24) on obtient l'expression (4.25) de l'estimée de la matrice de $\mathbf{G}$. En général, un vecteur de balayage théorique et un expérimental suffit pour calculer la matrice de calibration $\mathbf{G}$.

Cette technique permet de compenser plusieurs sources d'erreurs telles que le déphasage, le déséquilibre I/Q, l'incertitude de la position des capteurs et le couplage mutuel, mais elle ne règle pas le problème de différence des amplitudes. Par conséquent, cette dernière devra être fixée avant l'utilisation de la matrice $\mathbf{G}$. D'autre part, le problème du couplage mutuel n'est pas observé dans notre cas de sorte qu'uniquement la diagonale de la matrice $\mathbf{G}$ est nécessaire pour la calibration des données.

Une fois $\hat{\mathbf{G}}$ calculée, la calibration doit être appliquée soit aux signaux I/Q (4.26) ou à la matrice de covariance (4.27) avant l'algorithme MUSIC. Cette matrice est valide tant que l'environnement dans lequel les mesures sont faites reste inchangé. Autrement, une nouvelle matrice $\mathbf{G}$ sera de mise.

$$\mathbf{Y} = \hat{\mathbf{G}}\mathbf{X} \tag{4.26}$$

$$\mathbf{R}_{yy} = \hat{\mathbf{G}}\mathbf{R}_{xx}\hat{\mathbf{G}}^{\dagger} \tag{4.27}$$

4.8 Conclusion

Ce chapitre a fait l'apanage de la présentation des équations mathématiques et méthodes possibles à la réalisation de projet. Tout d'abord, des précisions sur les caractéristiques propres à la matrice de covariance $\mathbf{R}_{xx}$ précédemment définie au chapitre 1 ont été apportées. En effet, grâce à ces propriétés hermitienne et symétrique, seulement la moitié de la matrice $\mathbf{R}_{xx}$ est calculée. Cela réduira l'allocation en ressources et réduira le temps de calcul du FPGA.

Puis, les équations mathématiques de la transformée unitaire et son inverse ont été présentées. Celle-ci se résume, dans le cas de la transformée directe, aux multiplications à gauche et à droite de la matrice complexe à convertir par la matrice unitaire $\mathbf{U}$. Dans le cas de la transformée inverse, une simple multiplication à gauche de la matrice complexe est de mise. Aussi, l'intérêt de cette transformée

unitaire est de réduire la charge de calcul de la décomposition EVD par le FPGA. Cette charge serait très importante avec une matrice de covariance complexe, voire même impossible à implémenter dans le FPGA utilisé dans le projet.

La méthode de Jacobi pour la décomposition EVD générale a été abordée. Puis, les détails du calcul des valeurs propres d'une part, et des vecteurs propres d'autre part, ont été discutés. Ces calculs sont itératifs et à chaque itération une double multiplication par une matrice de rotation **S** est appliquée à la matrice partielle (matrice à diagonaliser) des valeurs propres dont la matrice de départ est la matrice de covariance. Puis une simple multiplication entre la matrice de rotation et la matrice partielle des vecteurs propres égalée. Cette dernière est initialement égale à la matrice identité, mais converge vers celle des vecteurs propres. De plus à la fin de chaque itération un réordonnancement est effectué afin de préparer l'itération suivante. Le nombre d'itérations théoriques est égal pour une matrice de covariance de taille $N \times N$ à $N \log N$ d'après la méthode de Jacobi, mais on verra au chapitre suivant qu'en pratique un ajustement est nécessaire.

Avant de donner des détails sur le calcul du pseudo-spectre, la méthode du tri à bulle est appliquée aux valeurs propres et leurs vecteurs propres associés. Ce tri est réalisé en vue de constituer une matrice formée par les vecteurs propres sources classés en ordre décroissant de leurs valeurs propres associées. Le nombre de vecteurs propres sources est déterminé sur la base d'un pourcentage de l'erreur moyen.

Enfin, il a été démontré dans ce chapitre, la nécessité de calibrer le réseau d'antennes et la technique pour le faire a été présentée. Cette technique utilise une matrice de calibration **G** calculée à partir du vecteur propre source principal et du vecteur **a** théorique pour un angle d'arrivée de référence. Ainsi, la calibration peut s'appliquer directement aux signaux bruts ou à la matrice de covariance par une multiplication de l'un ou l'autre avec la matrice **G**.

Chapitre 5

Implémentation du système proposé dans un FPGA

5.1 Introduction

Après avoir établi les fondements théoriques, il est temps d'expliquer l'architecture retenue à son implémentation sur FPGA. Le système proposé est représenté par un ensemble de sous-systèmes ou blocs de traitement numérique. De cette façon, la complexité du système est divisée et répartie entre les sous-systèmes de complexité moindre, facilitant ainsi la lisibilité et le débogage durant le développement du code. Les sections de ce chapitre s'appuient sur ce découpage pour expliquer le fonctionnement de chaque bloc de traitement numérique en s'aidant de schémas ou directement d'images du bloc d'implémentation du code *System generator*.

La figure 5.1 montre la chaine de traitement numérique du système dans son ensemble. Elle débute par l'acquisition des échantillons de données à 125 MHz avec une résolution de 14 bits représentant les caractéristiques de la carte FMC d'acquisition, suivie d'un pré traitement des données (filtrage numérique et normalisation) prérequis à l'obtention de la matrice de covariance. Ensuite, si une matrice de calibration **G** est disponible et valide pour l'environnement de mesures du système, l'implémentation de l'algorithme MUSIC débute selon le processus qui suit : décomposition en valeurs et vecteurs propres après la transformation unitaire pour passer d'une matrice de covariance complexe à réelle ; tri en ordre décroissant des vecteurs et valeurs propres ; détection du nombre de sources en vue d'identifier les vecteurs propres sources utilisés pour le calcul du pseudo-spectre MUSIC. Sinon, la suite de traitement sert à inclure le calcul d'une nouvelle matrice **G** avant calibration : on commence par placer une source à 90° par rapport à l'axe du réseau non calibré ; transformation unitaire de la matrice de covariance résultante pour cette source de référence à 90° ; décomposition en valeurs et vecteurs propres ; tri ; utilisation du vecteur source et du vecteur d'analyse théorique à 90° pour déterminer la matrice **G** désirée.

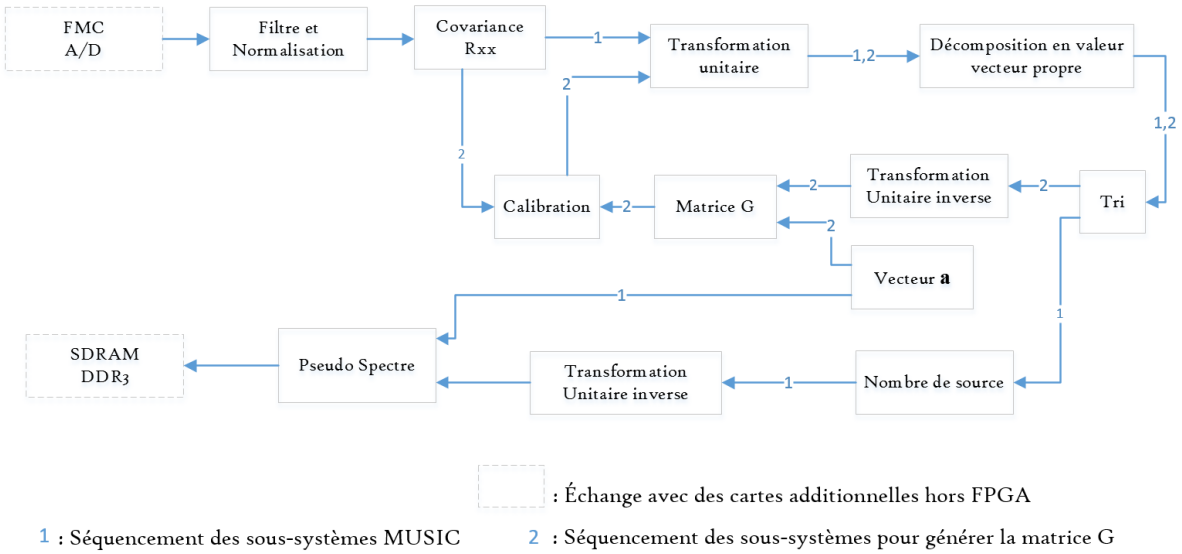


FIGURE 5.1 – Séquence de l'ensemble des sous-systèmes pour l'implémentation de MUSIC et la calibration de données

5.2 Acquisition

L'acquisition est réalisée par la carte MI-125 FMC installée dans le *PicoDigitizer* de la série 125. L'approche MBDK offre une palette d'outils permettant de cibler et d'utiliser les cartes composant le *PicoDigitizer*. Le block 125 fait partie de ces outils. Il est utilisé dans *Simulink* et *System generator* pour permettre l'accès aux fonctionnalitées de la carte MI-125, notamment la réception des signaux provenant des convertisseurs analogiques numériques (A/D).

Dans le cadre du projet, les signaux à analyser sont au nombre de 16 (8 canaux scindés en phase et quadrature). Un nombre de 16 blocs est alors nécessaire pour acquérir les signaux. Ces blocs sont configurés comme suit :

- Type : MI125-16, car une seule carte MI125-16 est connectée à carte-mère *Perseus*.
- Mode : ADC, pour accéder directement aux signaux numériques convertis par les convertisseurs A/D.
- Channel ID : $1 \cdots 16$, identifie les canaux. Les canaux paires représentent les signaux en phase (I).

Le format de sortie des signaux est signé 14 bits et fixé par la résolution des convertisseurs A/D avec une fréquence d'échantillonnage f_s de 125 Mbit/s. La figure 5.2 illustre le bloc MI125-16.

5.2.1 Filtre numérique

La figure 5.3 illustre un signal brut acquis directement des convertisseurs A/D sans traitement préalable. On peut observer à partir de ce signal brut qu'il résulte de la somme de deux signaux. Le premier est le

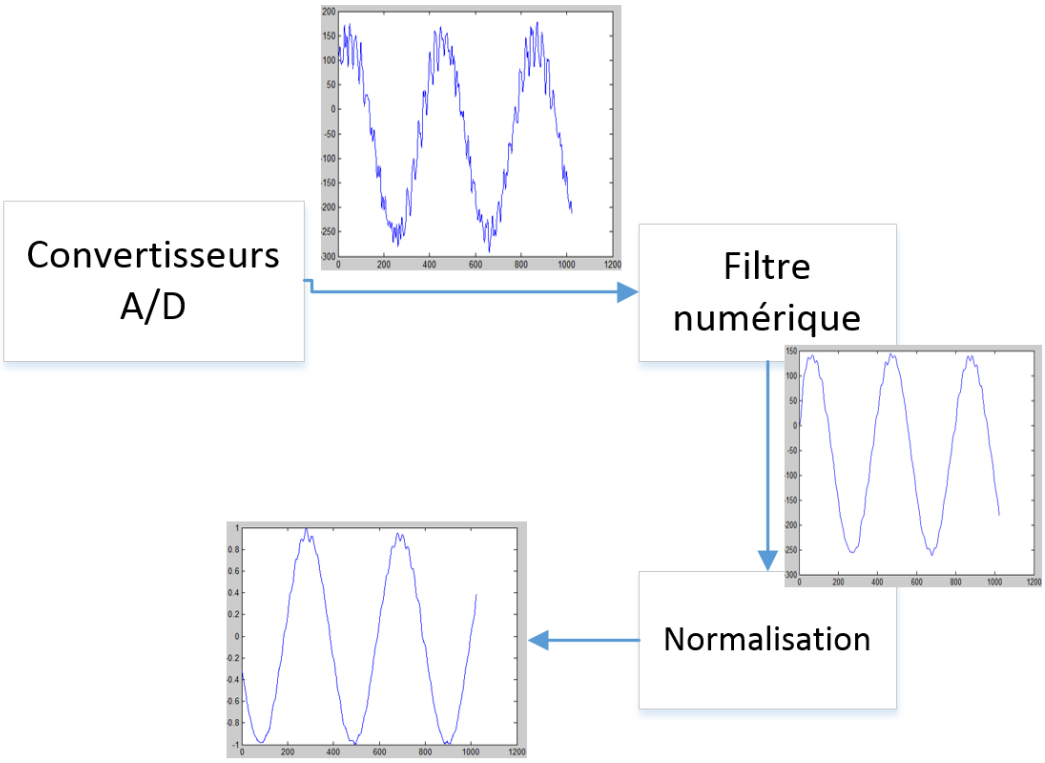


FIGURE 5.2 – Montage pour l'acquisition des signaux

signal désiré de basse fréquence (quelques centaines de kilo Hertz) et le second signal perturbateur de haute fréquence (quelques dizaines de Méga Hertz). Ce second signal provenant des harmoniques non filtrées par la carte électronique de modulation, induit des erreurs dans le calcul des angles d'arrivée par MUSIC. Il est donc nécessaire de l'éliminer avant l'exécution de MUSIC. La solution adoptée pour éliminer cette source non désirée est d'utiliser un filtre passe-bas numérique.

Le bloc filtre réalisé dans le projet est basé sur le *FIR compiler 6.3* de *Xilinx*, bloc qu'on retrouve dans la bibliothèque *AXI4* et *DSP* de l'environnement *Simulink*. Le *FIR compiler 6.3* permet de générer un filtre à réponse impulsionnelle finie (FIR) de haute performance, efficient pour les FGPAs et hautement paramétrisable. Plus d'informations sur ce bloc peuvent être trouvées dans [18]. Les principales spécifications de ce filtre passe-bas qui sont utilisées comme paramètres dans le *bloc FIR compiler 6.3* ont été calculées par l'outil *FDATool*. Cet outil fournit une interface utilisateur permettant de concevoir et d'analyser rapidement un filtre. La référence [7] apporte plus d'informations sur cet outil. La figure 5.4 montre les caractéristiques du filtre réalisé via *FDATool*.

Pour filtrer idéalement les 16 signaux bruts acquis, 16 filtres montés en parallèle sont requis. Dans la pratique, ce nombre est très coûteux en termes d'espace de calcul pour le *Virtex 6* utilisé. C'est pourquoi uniquement 8 filtres sont utilisés pour traiter les signaux, causant une perte d'un échantillon sur deux. En effet, dans cette configuration, un filtre est dédié à la fois pour le I et le Q d'un même canal. Ainsi, à un instant t1, il filtre un échantillon du I, laissant celui du Q, puis un coup d'horloge plus

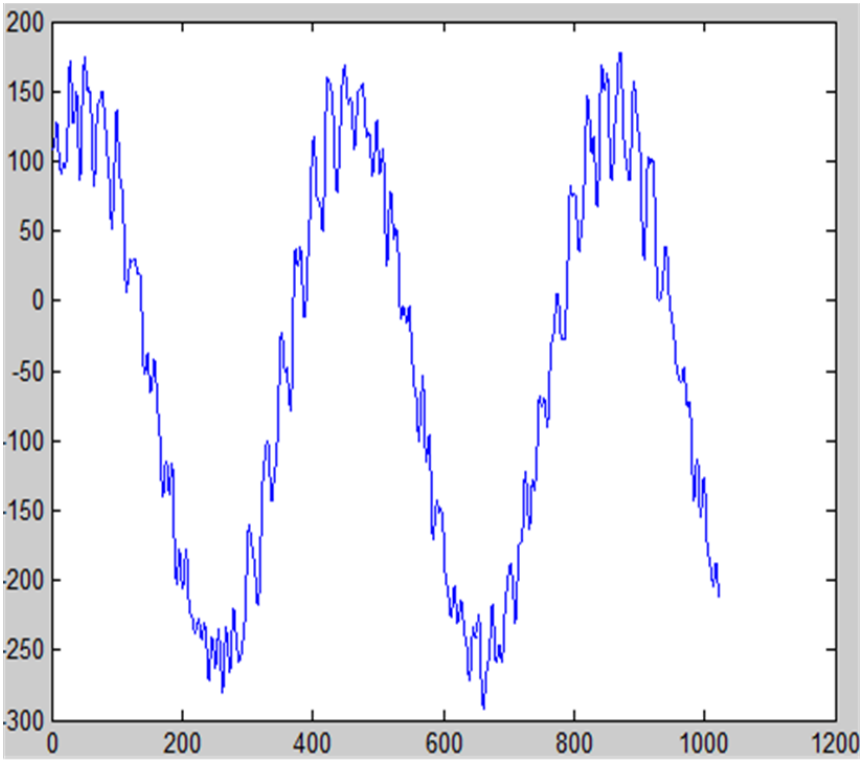


FIGURE 5.3 – Extrait du signal en phase (I) du canal 7 avant le filtre passe-bas

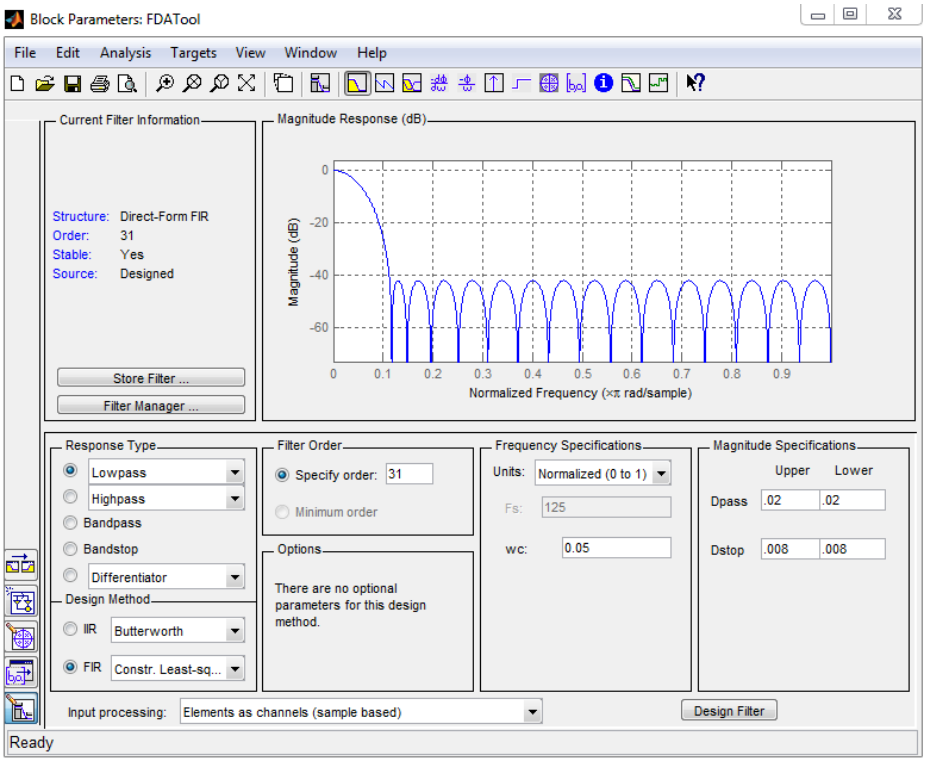


FIGURE 5.4 – Modèle du filtre passe-bas conçu par *FDATool*

tard filtre celui du Q en laissant celui du I tel quel cette fois. Après validation, ceci n'a pas d'impact significatif sur le résultat de MUSIC. La figure 5.5 montre le résultat obtenu après l'application du filtre au signal de la figure 5.3.

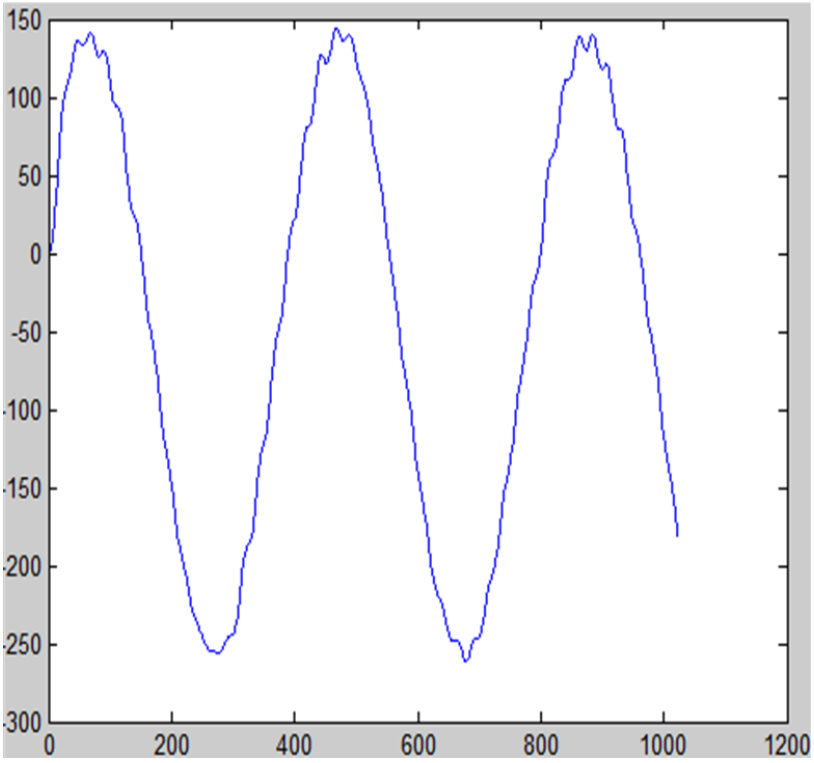


FIGURE 5.5 – Extrait du signal en phase (I) du canal 7 après le filtre passe-bas

5.2.2 Bloc Normalisation

Le bloc de normalisation revient à modifier l'amplitude de tous les signaux jusqu'à un seuil maximal de 1 et minimal de -1. Son but est double : favoriser le formatage des signaux afin qu'ils rencontrent les exigences des paramètres des données d'entrée du bloc *Cordic* utilisé pour la décomposition en valeurs propres et corriger le problème des différences de niveau des amplitudes des signaux I et Q.

Son fonctionnement s'effectue en deux étapes. Premièremenent, les valeurs *min* et *max* de chaque signal sont déterminées après lecture des premiers 1524 (choix arbitraire, représentant trois périodes du signal environ) échantillons faisant suite à chaque remise à zéro ou redémarrage du système. Dans la seconde étape, chaque signal est ajusté par la résultante de la différence entre ses valeurs *min* et *max* avant d'être divisé par sa valeur maximale. L'implémentation du bloc de normalisation est montrée aux figures 5.6 et 5.7. Plus de détails sont apportés à l'annexe C.1.

La réalisation d'un banc de test recevant en entrée le signal réel en phase du canal 7 (I-14) filtré a permis de valider le bloc de normalisation avant de l'incorporer dans le montage final. La figure 5.8 présente l'exemple d'un signal normalisé observé à la sortie du banc d'essai du bloc de normalisation.

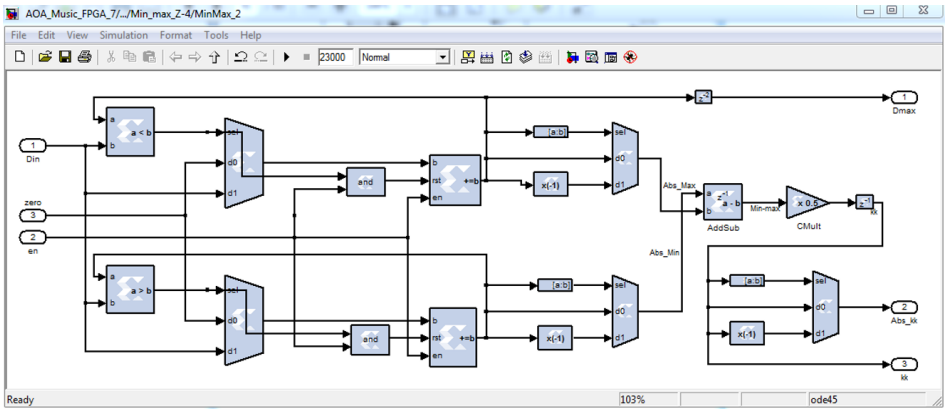


FIGURE 5.6 – Calcul des valeurs *min-max* avec les blocs *Xilinx*

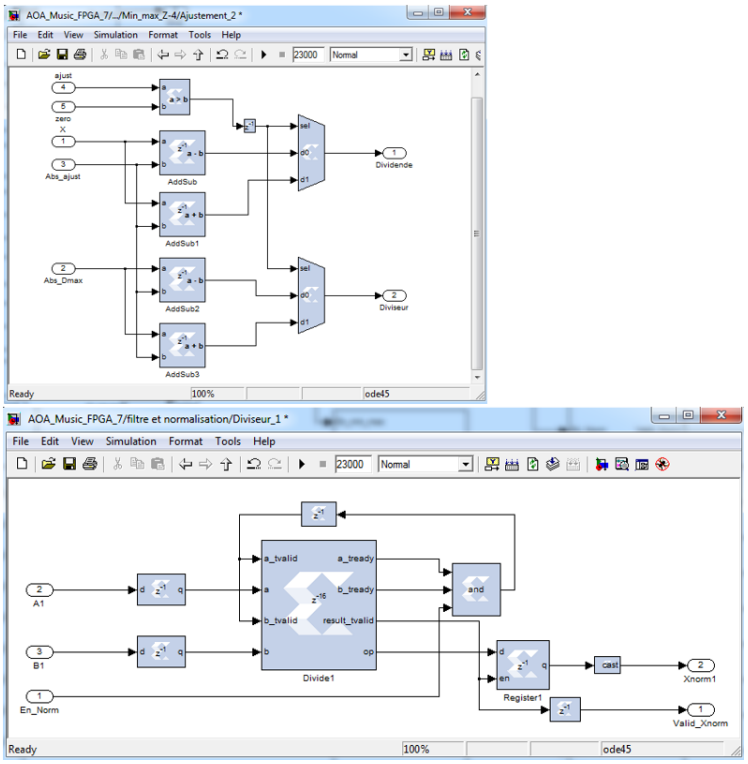


FIGURE 5.7 – Les sous-blocs *Ajustement* (au-dessus) et *Division* par la valeur *max* implémentée avec les blocs Xilinx

5.3 Bloc Matrice covariance

La formation de la matrice de covariance précédemment définie est un préalable à sa décomposition en valeurs et vecteurs propres. Son implémentation exploite la symétrie qui est une caractéristique intrinsèque de cette matrice. En effet, sur la base de cette symétrie, la partie triangulaire supérieure de la matrice de covariance est calculée et la seconde est déduite par simple changement de signe de la partie imaginaire de ses éléments. La figure 5.9 illustre l'architecture adoptée pour la formation de la matrice de covariance à partir des signaux reçus. Ces différentes parties sont expliquées ci-dessous.

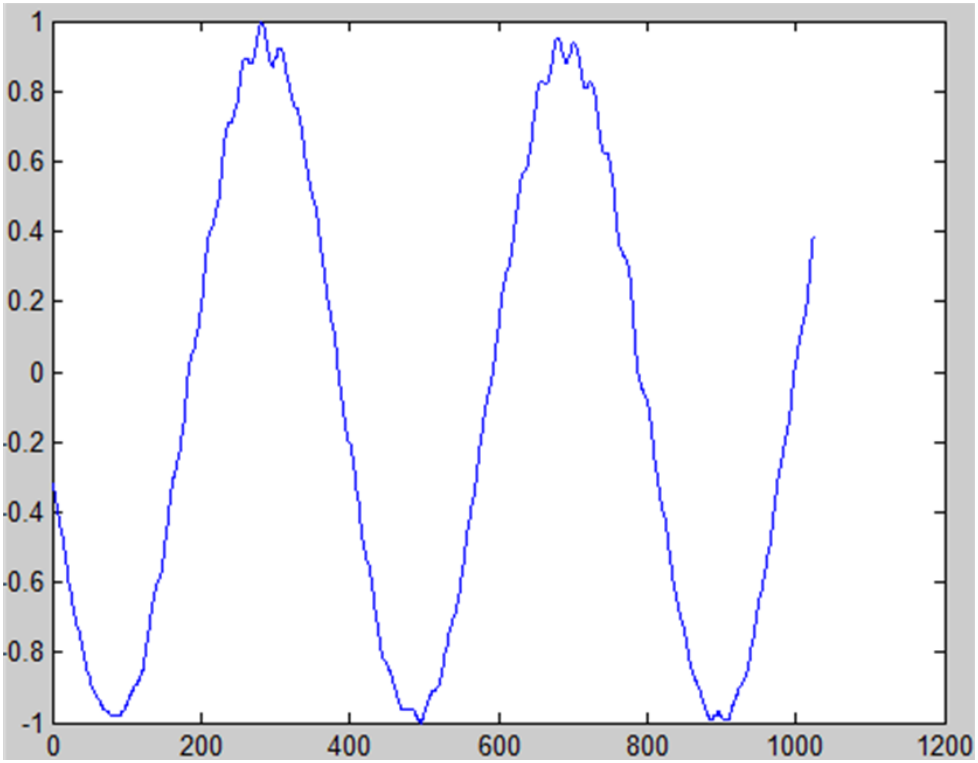


FIGURE 5.8 – Exemple de signal normalisé

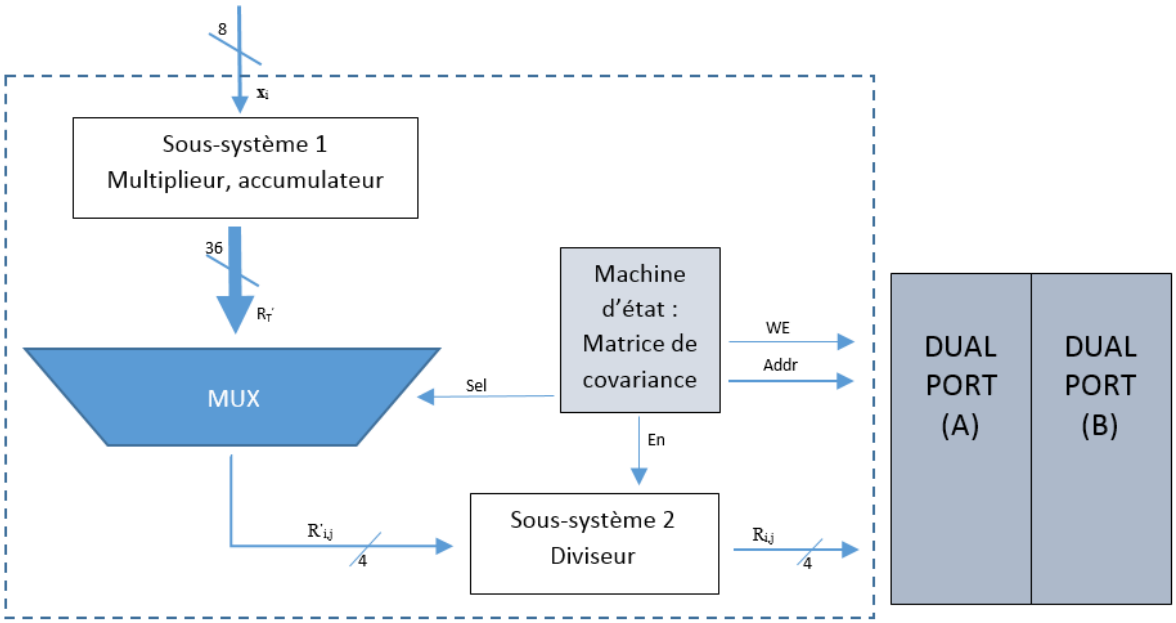


FIGURE 5.9 – Architecture pour le calcul de la matrice de covariance

— *Sous-système 1 :* Le sous-système reçoit, à chaque instant (kT), un vecteur complexe $\mathbf{x}_k$ de taille 1×8 représentant une épreuve des signaux I et Q, qu'il multiplie par sa transposée et additionne le résultat à celui obtenu précédemment à l'instant $(k-1)T$. L'opération est répétée

jusqu'au nombre d'épreuves M souhaité. Ce sous-système contient 28 sous-systèmes non-diagonaux et 8 diagonaux opérants en parallèle et destiné à calculer les 36 éléments de la matrice triangulaire supérieure de la matrice de covariance. La figure 5.10 illustre les sous-systèmes non-diagonaux et diagonaux qui sont composés de multiplieurs, additionneurs et accumulateurs. Aussi, l'agencement des arguments présenté à l'entrée de ces sous-systèmes est représenté par la figure 5.11.

— *MUX :* Sert à sélectionner 4 éléments parmi ceux de la matrice triangulaire de la matrice de covariance calculée non divisée encore par le nombre d'épreuves. La sélection de ce multiplexeur est effectuée par une machine à état établie en *m-code*. La machine à état établie par *m-code* assure adéquatement la sélection du MUX afin de reconstituer la matrice de covariance complète par séquence de sous-matrices de taille 2×2.

— *Sous-système 2 :* Assure une division simultanée de la sous-matrice de 4 éléments présentée au fur et à mesure à son entrée par le nombre d'épreuves M. Une fois l'opération terminée, les différentes sous-matrices de covariance obtenue sont transmises par ordre d'entrée dans le sous-système 2 aux mémoires Dual-Port (DP) A et B.

— *Machine à état (FSM) de la matrice de covariance :* Gère la sélection du bloc de multiplexeurs, assure des divisions valides et génère l'adresse de l'emplacement des éléments de la matrice de covariance dans la mémoire ainsi que le signal mettant la mémoire dans l'état d'écriture.

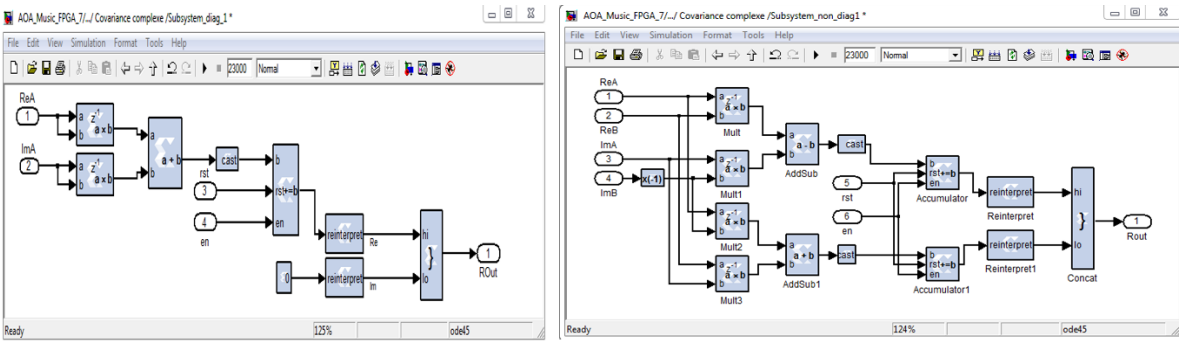


FIGURE 5.10 – Sous-systèmes diagonaux et non-diagonaux pour le calcul de la matrice de covariance

5.4 Bloc de la transformée unitaire et son inverse

La figure 5.12 illustre le bloc fonctionnel de l'implémentation de la transformée unitaire aussi bien que la transformée inverse. En effet, ce bloc effectue 2 transformations : la transformée unitaire, où il transforme la matrice de covariance complexe en matrice réelle ; la transformée unitaire inverse, où il transforme les vecteurs propres réels en vecteurs propres complexes. Dans le cas de la première transformation, les mémoires A et B contiennent initialement les éléments de la matrice de covariance complexe. Chaque case mémoire comprend une série de 32 bits issue de la concaténation de 16 bits de la partie réelle et 16 autres bits de la partie imaginaire d'un élément de la matrice de covariance. Dans le cas du calcul de la transformée inverse, qui survient après la décomposition EVD, les DPs A et B

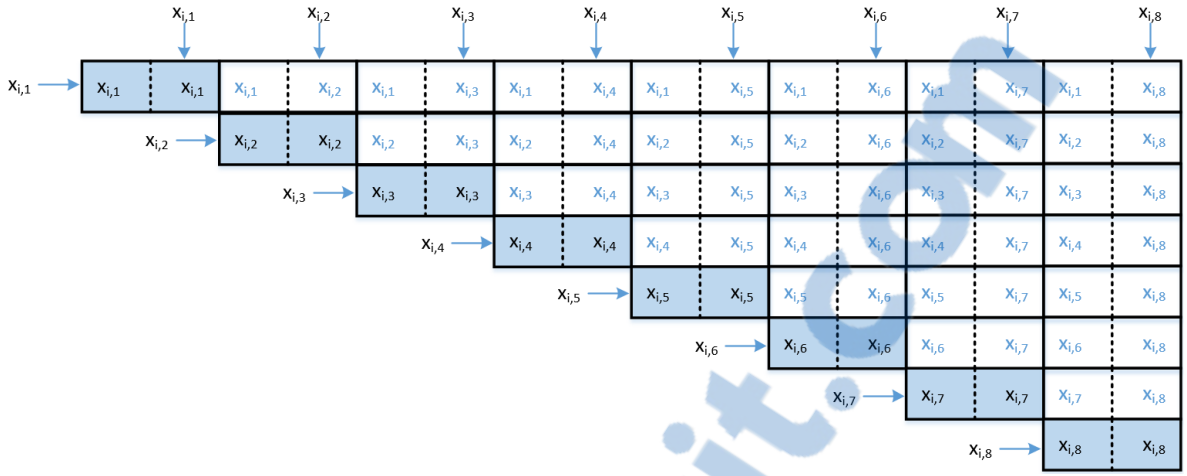


FIGURE 5.11 – Séquence de calcul de la moitié de la matrice de covariance à l'instant t

contiennent les matrices des valeurs et vecteurs propres respectivement **D** et **V**. Chaque case mémorise la résultante de 32 bits de la concaténation entre les éléments de même indice des matrices **D** et **V**. En d'autres termes, chacune des cases de DPs A et B est composée de 16 bits de $\mathbf{d}_{ij}$ et 16 bits de $\mathbf{v}_{ij}$.

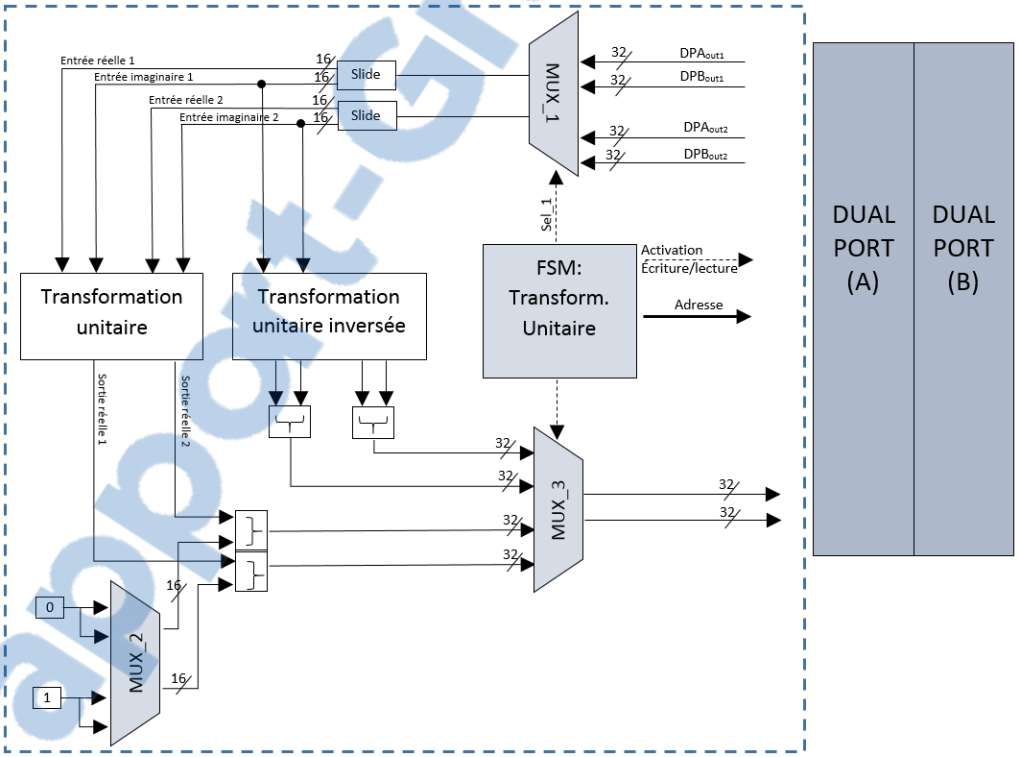


FIGURE 5.12 – Bloc fonctionel de la transformée unitaire et son inverse

Tout d'abord, la FSM génère les adresses correspondantes à l'emplacement des éléments de la matrice à transformer dans les DPs A et B. Les données de 32 bits lues aux 4 sorties des DPs sont ensuite multiplexés afin de choisir convenablement deux opérandes pour le calcul d'un élément de la matrice

transformée recherchée. Avant la transformation, chacune des deux données présentes à la sortie du *multiplexeur 1* voit sa trame binaire scindée de moitié afin de dissocier les deux valeurs préalablement concaténées. Ainsi, dans le cas de la transformée unitaire, il s'agit des parties réelles et imaginaires des deux opérandes du bloc identifié Transformation unitaire à la figure 5.12. Pour ce qui est du second bloc identifié Transformation unitaire inverse, il s'agit de 2 éléments des vecteurs propres réels.

Ensuite, les 2 opérandes dans leur nouvelle forme sont transmis aux blocs de transformations. Ces blocs représentent le modèle d'implémentation des équations (4.1) et (4.6) où les matrices à transformer sont multipliées une ou deux fois selon le cas, par la matrice $\mathbf{U}$ (voir 4.2). Cette dernière, est essentiellement composée de $1, -1, j, -j$ et 0, des quantités (valeurs) qui ne modifient pas l'amplitude, mais plutôt change le signe ou intervertit parties réelle et imaginaire de la résultante de l'opération de multiplication. C'est pourquoi ces blocs sont composés d'inverseurs de signe dans les deux cas de transformations et d'opérateurs d'additions dans le cas du bloc de la transformée unitaire. La figure 5.13 donne un aperçu de ces blocs.

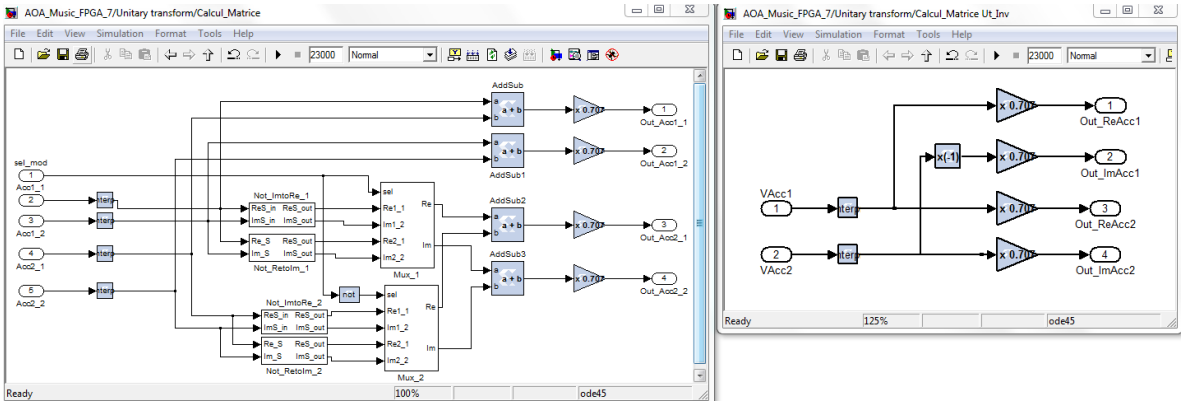


FIGURE 5.13 – Modèle des blocs de la transformée unitaire et son inverse

Grâce à la symétrie des nouvelles matrices recherchées, chaque bloc de transformations renvoie la valeur transformée et sa conjuguée. De cette manière, le temps de traitement est réduit environ de moitié. À la fin du calcul, les sorties de chaque bloc de transformation sont formatées en 32 bits avant d'être sélectionnées convenablement par la machine à état via le *multiplexeur 3* et enregistrées dans les mémoires A et B. Le formatage, dans le cas de la transformée unitaire, consiste à concaténer les 2 éléments réels : soit un élément de 16 bits de la matrice de covariance transformée avec un élément (un 0 ou un 1) sur 16 bits appartenant à la matrice identité servant de matrice de départ à l'obtention de la matrice des vecteurs propres à obtenir (voir sous section 4.4.2). Dans le cas de la transformée inverse, le formatage consiste à concaténer la partie réelle et la partie imaginaire, de 16 bits chacune pour les 2 éléments de la nouvelle matrice des vecteurs propres complexes calculée.

5.5 Bloc de la décomposition en valeurs et vecteurs propres

Le bloc fonctionnel de la décomposition en valeurs et vecteurs propres vue au chapitre précédent est présenté à la figure (5.17). Son fonctionnement se résume en trois opérations majeures : calcul des angles par les processeurs diagonaux, rotation double aux processeurs des valeurs propres et rotation simple aux processeurs des vecteurs propres, et finalement réordonnancement. À ce stade, la matrice de covariance transformée en réelle est sauvegardée de sorte que ses lignes paires sont dans la mémoire DP A et les impaires dans la DP B. Chacun des éléments de cette matrice de covariance est concaténé à un élément de la matrice de départ des vecteurs propres (matrice identité). Par exemple l'adresse 0 du DP A renferme le premier élément sur la première ligne de la matrice de covariance réelle et celui de la matrice des vecteurs. Aussi, il est important d'avoir en tête le concept des sous-matrices décrites au chapitre 4, car elles sont formées par les quatre sorties des DPs A et B.

5.5.1 Bloc *angles*

Le calcul des angles est la première opération à la décomposition en valeurs et vecteurs propres. Son implémentation repose sur l'équation (4.12) et est illustrée à la figure 5.14. Premièrement, les sous-matrices 2×2 diagonales sont lues depuis les DPs A et B. Ensuite, le numérateur et le dénominateur de la fraction opérande de l'arc-tangente sont calculés respectivement et transmis aux entrées *Xin* et *Yin* du *Cordic*. Ce dernier, une fois les données à son entrée validée, calcule en mode pipeline l'arc-tangente, ainsi après un délai équivalent au nombre de bits des données plus 2, transmet le résultat disponible à sa sortie. L'angle obtenu est divisé en deux par un décalage d'un bit à droite. L'opération est répétée avec les sous-processeurs diagonaux jusqu'à atteindre $N/2$ angles.

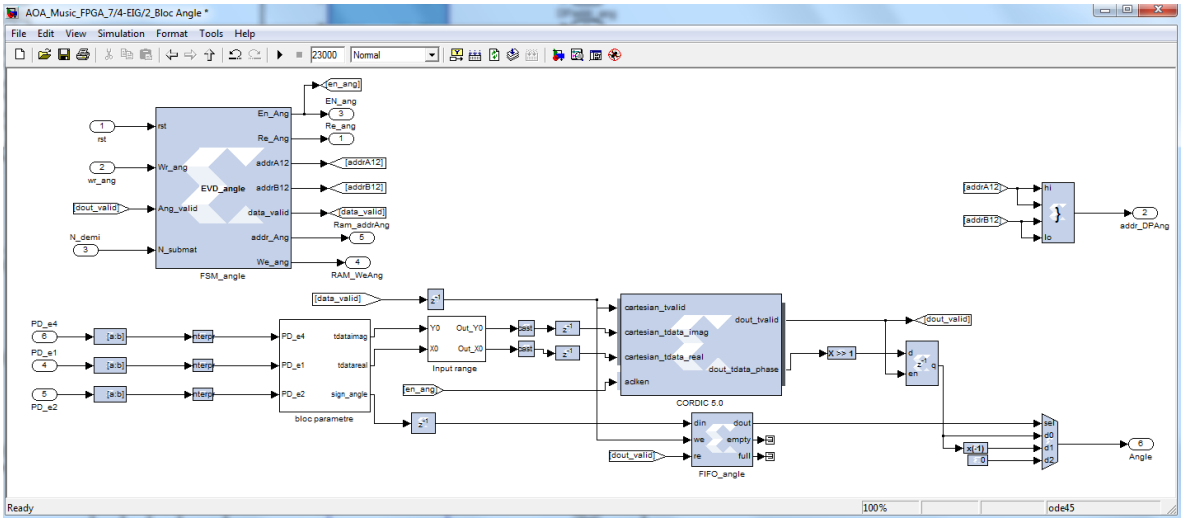


FIGURE 5.14 – Bloc d'implémentation du calcul des angles de rotation

L'algorithme du *Cordic* résout l'opération d'arc-tangente par le mode de fonctionnement vectoriel avec des coordonnées circulaires. En d'autres termes, le calcul de $\theta_{i,i}^{(h)}$ n'est autre que l'angle résultant de la rotation d'un vecteur de coordonnés $[r_{2l,2m}^{(h)} - r_{2l-1,2m-1}^{(h)}, 2r_{2l-1,2m}^{(h)}]$, comme le montre la figure 5.15. Le

bloc *Cordic* de *Xilinx version 6.0* impose une notation 1Q*n* des données à son entrée, c'est-à-dire 1 bit de signe, 1 bit à gauche du point décimal et *n* bits de fraction. Quant aux angles, ils sont à virgule fixe sous la forme 2Q*n*, c'est-à-dire 1 bit de signe, 2 bits d'entier et *n* bits de fraction.

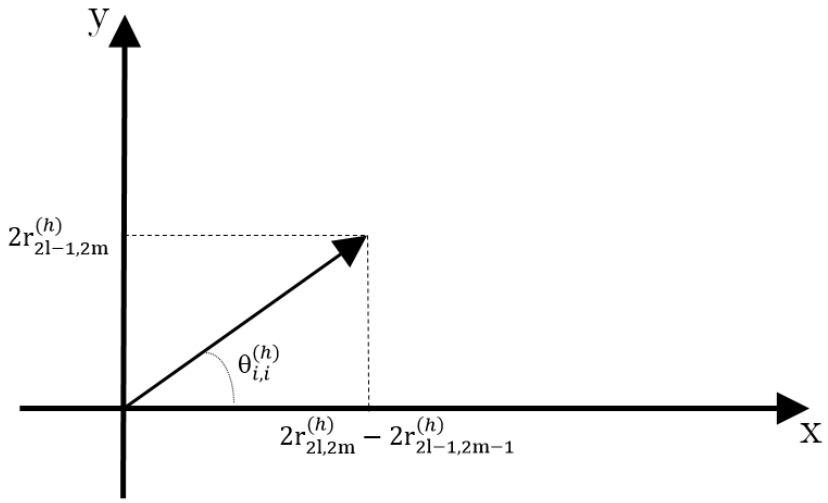


FIGURE 5.15 – Représentation de la rotation du vecteur de coordonnées $[r^{(h)}_{2l,2m} - r^{(h)}_{2l-1,2m-1}, 2r^{(h)}_{2l-1,2m}]$ pour l'obtention de l'angle $\theta^{(h)}_{i,i}$

Il est important de noter que les angles calculés doivent être compris dans la plage $[-\frac{\pi}{2}; +\frac{\pi}{2}]$, comme il est le cas de la fonction *atan* de *Matlab*. Cela afin d'éviter toute surcharge des données plus tard aux entrées des *Cordics* de rotations et par la même occasion réduire la charge de calcul dans le FPGA. Le *Cordic* en mode arc-tangente cependant opère sur le cercle trigonométrique au complet lorsque le signe du numérateur présenté à son entrée *Xin* (entrée du numérateur) est négatif. Plus précisément, les phases retournées par le *Cordic* dans ce cas précis sont étendues sur la plage d'angles $[-\pi; +\pi]$ ce qui est équivalent à la fonction *atan2* de *Matlab*. Par exemple pour une combinaison à l'entrée du *Cordic Xin* $= -0.6250$ et *Yin*$= +0.5$, l'angle obtenu est $+2.4669$ au lieu de -0.6747 obtenu par *atan*. De même, en gardant la même combinaison et changeant uniquement le signe du dénominateur de sorte que *Yin*$= -0.5$, le résultat obtenu est le même, mais de signe différent -2.4669. Ceci parce que, lorsque l'entrée du *Cordic Xin* est négative, ce dernier met son résultat dans le second quadrant. Pour remédier à ce problème, un circuit d'ajustement du signe des angles calculés est ajouté au montage. Ce dernier procède premièrement à la détection successive du signe des paramètres à l'entrée du numérateur du *Cordic*. Ensuite, il les rend positifs s'ils sont négatifs et fait suivre l'information sur leur signe exact afin de les synchroniser avec la sortie du *Cordic*. Enfin, le circuit d'ajustement corrige l'angle fourni de 180 degrés si nécessaire. Les angles ainsi ajustés sont transmis et enregistrés dans une RAM.

5.5.2 Bloc *rotation*

Comme présenté au chapitre des solutions théoriques, la décomposition en valeurs et vecteurs propres d'une matrice de covariance passe par sa double multiplication avec une matrice de rotation **S**. En

réécrivant l'équation (4.16) de la double rotation en l'équation (5.1) et considérant seulement la première multiplication $\mathbf{Q}_{i,j}^{(h)}$ exprimée par (5.2) alors l'équation (5.1) devient (5.3).

$$\begin{bmatrix} r_{2l-1,2m-1}^{(h+1)} & r_{2l-1,2m}^{(h+1)} \\ r_{2l,2m-1}^{(h+1)} & r_{2l,2m}^{(h+1)} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{i,i}^{(h)}) & \sin(\theta_{i,i}^{(h)}) \\ -\sin(\theta_{i,i}^{(h)}) & \cos(\theta_{i,i}^{(h)}) \end{bmatrix}^{\top} \begin{bmatrix} s_{2l-1,2m-1}^{(h+1)} & r_{2l-1,2m}^{(h+1)} \\ s_{2l,2m-1}^{(h+1)} & s_{2l,2m}^{(h+1)} \end{bmatrix} \begin{bmatrix} \cos(\theta_{j,j}^{(h)}) & \sin(\theta_{j,j}^{(h)}) \\ -\sin(\theta_{j,j}^{(h)}) & \cos(\theta_{j,j}^{(h)}) \end{bmatrix} \tag{5.1}$$

$$\mathbf{Q}_{ij}^{(h)} = \begin{bmatrix} \cos(\theta_{i,i}^{(h)}) r_{2l-1,2m-1}^{(h)} - \sin(\theta_{i,i}^{(h)}) r_{2l,2m-1}^{(h)} & \cos(\theta_{i,i}^{(h)}) r_{2l-1,2m}^{(h)} - \sin(\theta_{i,i}^{(h)}) r_{2l,2m}^{(h)} \\ \sin(\theta_{i,i}^{(h)}) r_{2l-1,2m-1}^{(h)} + \cos(\theta_{i,i}^{(h)}) r_{2l,2m-1}^{(h)} & \sin(\theta_{i,i}^{(h)}) r_{2l-1,2m}^{(h)} + \cos(\theta_{i,i}^{(h)}) r_{2l,2m}^{(h)} \end{bmatrix} \tag{5.2}$$

$$\mathbf{R}_{ij}^{(h+1)} = \mathbf{Q}_{ij}^{(h)} \begin{bmatrix} \cos(\theta_{j,j}^{(h)}) & \sin(\theta_{j,j}^{(h)}) \\ -\sin(\theta_{j,j}^{(h)}) & \cos(\theta_{j,j}^{(h)}) \end{bmatrix} \tag{5.3}$$

On peut en observer qu'elle équivaut à une rotation d'un vecteur de coordonnées $[r_{2l-1,2m-1}^{(h)}, r_{2l,2m-1}^{(h)}]$ par un angle $\theta_{i,i}$ et la rotation du vecteur de coordonnées $[r_{2l-1,2m}^{(h)}, r_{2l,2m}^{(h)}]$ par le même angle $\theta_{i,i}$. De ce fait, le *Cordic* de *Xilinx* en mode rotation sera une fois de plus l'outil approprié pour l'implémentation des simples et doubles rotations des processeurs vectoriels diagonaux PD et non diagonaux PND respectivement.

En mode rotation, le *Cordic* possède trois entrées de données. Les entrées *Xin* et *Yin* acceptent la notation 1Q*n* utilisée pour renseigner respectivement les abscisses et ordonnées des vecteurs à pivoter et une entrée *Yin* qui accepte les angles de rotation en notation 2Q*n*. Puisque les signaux non calibrés ont été normalisés avant le calcul de la matrice de covariance, les éléments de celle-ci respectent la notation 1Q*n*. Par contre, après la calibration de la matrice de covariance, les éléments calibrés obtenus ne respectent pas le formatage 1Q*n*. Un bloc de formatage avant chaque *Cordic* et de formatage inverse à leur sortie a été réalisé. Plus précisément, les coordonnées à l'entrée de chaque *Cordic* sont ramenées chacun au format de 16 bits signés avec 1 bit de signe, 1 bit pour la partie entière et 14 bits de fraction. Contrairement aux coordonnées, les angles respectent le format 2Q*n* puisqu'ils ont été préalablement calculés par un module *Cordic*. Les autres entrées du *Cordic* sont utilisées pour le contrôle du flux et la validation des données. De l'entrée des données à la sortie du *Cordic*, il s'écoule un temps correspondant au nombre de bits des données d'entrées plus 2 coups d'horloge et les vecteurs à la sortie du *Cordic* sont formatés 1Q*n*.

Dans l'architecture implémentée, les rotations doubles et simples des processeurs se partagent 2 *Cordics* de *Xilinx* opérant en parallèle. Afin d'obtenir le résultat escompté, les éléments des différents processeurs à pivoter sont assignés adéquatement aux entrées des *Cordics*. Par exemple, dans le cadre de la double rotation, pour assurer la seconde rotation à savoir la multiplication du résultat de la

première multiplication par la matrice de rotation conjuguée, l'ordonnée de la sous-matrice résultante de la première rotation du *Cordic A1* est assignée à l'entrée abscisse *Xin* du *Cordic A2*, et la sortie abscisse obtenue suite à la première rotation du *Cordic2* est assignée à l'entrée ordonnée du *Cordic A1*.

5.5.3 Ordonnancement

La troisième opération du sous-système dédié à la décomposition en valeurs et vecteurs propres est constituée de deux ordonnancements. Ces derniers opèrent après l'opération de rotation. Ils sont appliqués aux processeurs PD et PND au fur et à mesure jusqu'à reconstitution des nouvelles matrices devant servir à la prochaine itération. Les ordonnancements sont effectués dans l'ordre suivant : échange entre les éléments à l'intérieur d'un même processeur dans un premier temps, ensuite le déplacement de ces mêmes éléments vers les processeurs voisins.

L'implémentation du bloc d'échange interne des éléments implique quatre multiplexeurs dont les sorties sont reliées aux entrées des mémoires DPs A et B. Les quatre entrées des multiplexeurs sont identiquement reliées aux nouvelles valeurs des processeurs. Ainsi, la FSM locale génère la combinaison adéquate des 4 signaux de sélection des multiplexeurs afin d'assurer la permutation illustrée à la figure 4.6. La figure 5.16 illustre le bloc implémentation développé avec le *System generator*.

D'autre part, le déplacement externe est assuré uniquement par la machine à état locale. Ce dernier génère les adresses des nouvelles positions des éléments dans les DPs A et B. Les adresses de chaque élément sont calculées à partir des équations (5.4) et (5.5) qui comprennent deux paramètres : i et j. Ces deux paramètres représentent les nouvelles coordonnées d'un élément dans les matrices soumises à la rotation.

$$\text{AddrDP}_{A1\_2} = 2((i-1)\frac{N}{2}+(j-1)) \tag{5.4}$$

$$\text{AddrDP}_{B1\_2} = 2((i-1)\frac{N}{2}+(j-1))+1. \tag{5.5}$$

5.5.4 Architecture complète de la décomposition EVD

La figure 5.17 illustre l'architecture de fonctionnement résumant l'implémentation de la décomposition EVD de la matrice de covariance. Elle est basée sur le principe du tableau systolique par traitement successif des sous-matrices 2×2. Ainsi, elle présente l'avantage d'être invariante à la taille de la matrice à décomposer. Sa structure interne est constituée principalement de trois blocs de *Cordic*, des multiplexeurs, des registres à décalage FIFOs et une mémoire RAM.

Primo, les angles à l'itération h sont calculés et sauvegardés dans la RAM interne au FPGA. Ensuite, les processeurs PD et PND de la matrice partielle des valeurs propres et leurs angles associés sont lus et transmis aux *Cordics A1* et *A2* pour une première rotation. Après la lecture des PD et PND, les processeurs PV de la matrice partielle des vecteurs propres et leurs angles associés sont lus à leur tour et envoyés aux *Cordics* de rotation. Lorsque les résultats de la rotation des PD et PND sont

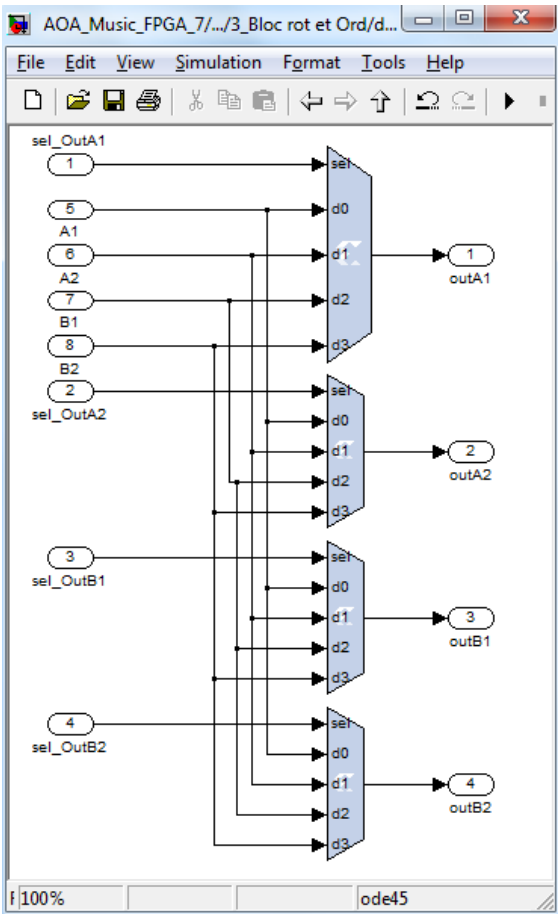


FIGURE 5.16 – Bloc d'implémentation de l'échange interne aux sous-matrices

rendus disponibles par les *Cordics*, ils sont transmis aux registres à décalage FIFOs A et B attendant la fin de la transmission des PVs aux *Cordics*. Lorsque les entrées des deux *Cordics* de rotation sont à nouveau disponibles, les résultats de la première rotation des PD et PND sauvegardés dans les registres à décalage FIFOs A et B leurs sont transmis. Les mêmes registres à décalage FIFOs sont ensuite utilisés pour sauvegarder temporairement les PVs pivotés jusqu'à la sortie des résultats de la seconde rotation des PD et PND. Enfin, les processeurs sont sauvegardés en mémoire après ordonnancement. Le processus est répété tant que le nombre d'itérations est inférieur à l'équation théorique donnée par Jacobi $N\log(N)$ plus 5 autres, une constante observée lors des manipulations, avec N le nombre de capteurs ou taille des matrices des vecteurs et valeurs propres.

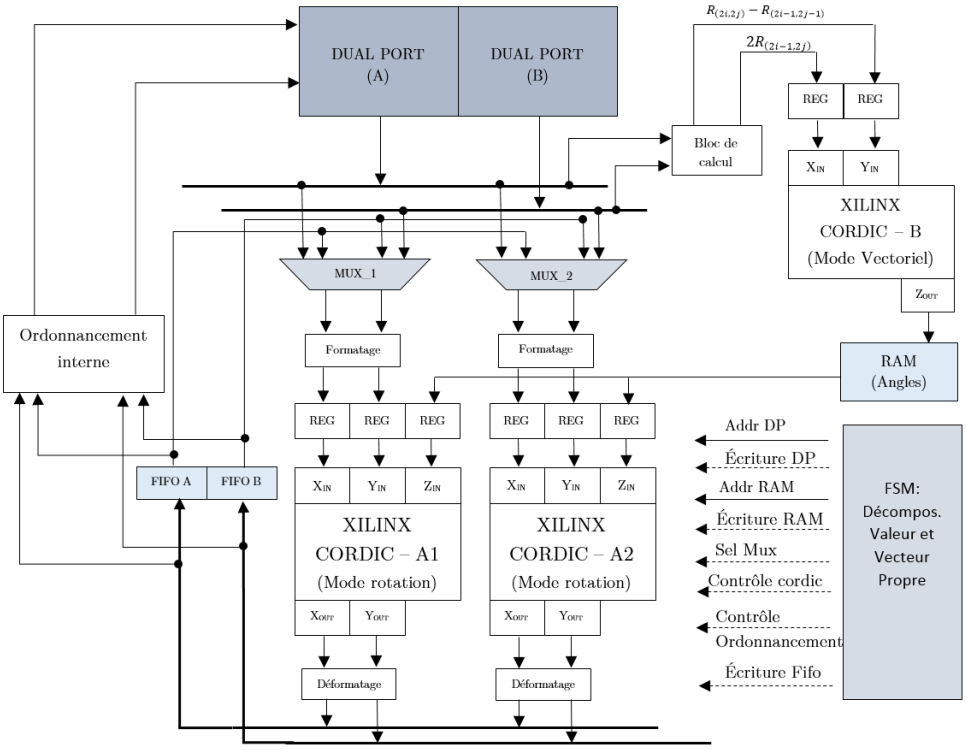


FIGURE 5.17 – Architecture de fonctionnement systolique pour la décomposition des valeurs et vecteurs propres

5.6 Bloc *tri* et détection du nombre de sources

5.6.1 Bloc *tri*

Le bloc d'implémentation du *tri* est activé à la fin de l'opération de décomposition en valeurs et vecteurs propres. À ce stade, les DPs A et B contiennent respectivement la matrice diagonale réelle contenant les valeurs propres et la matrice des vecteurs propres. Les éléments des deux matrices de 16 bits chacuns sont concaténés et enregistrés à la même adresse mémoire. Aussi, la configuration fait de sorte que la DP A contient les colonnes impaires des 2 matrices et les colonnes paires sont contenues dans la DP B.

La figure 5.18 illustre la structure interne du bloc d'implémentation effectuant le tri basé sur la technique à bulles en ordre décroissant. Tout d'abord, l'unité de contrôle du *tri* adresse les DPs A et B de façon à lire deux valeurs propres à la fois, dont une référence. Ces valeurs propres sont extraites des sorties des DPs A et B avant d'être comparées. Ensuite, en fonction du résultat de la comparaison transmis à l'unité de contrôle du *tri*, celle-ci permute ou pas les colonnes contenant les 2 valeurs propres en question dans le test et leurs vecteurs propres associés. Cette permutation est réalisée tout en conservant la diagonalisation de la matrice des valeurs propres. Par exemple, soit une matrice diagonale **D** contenant les valeurs propres et une matrice **V** contenant leurs vecteurs propres associés de taille 4×4. Chaque élément de la matrice **D** a été concaténé à l'élément de même indice de la matrice **V** afin de partager la même case mémoire. Si la valeur propre la plus significative se trouve à la colonne et à la ligne 4. En

effectuant uniquement les permutations entre les colonnes 1 et 4, pour respecter l'ordre décroissant, la valeur propre la plus significative se situerait désormais à la colonne 1 et à la ligne 4, ce qui détruirait la forme diagonale de la matrice **D**. Il faut donc réajuster les indices au niveau des lignes aussi.

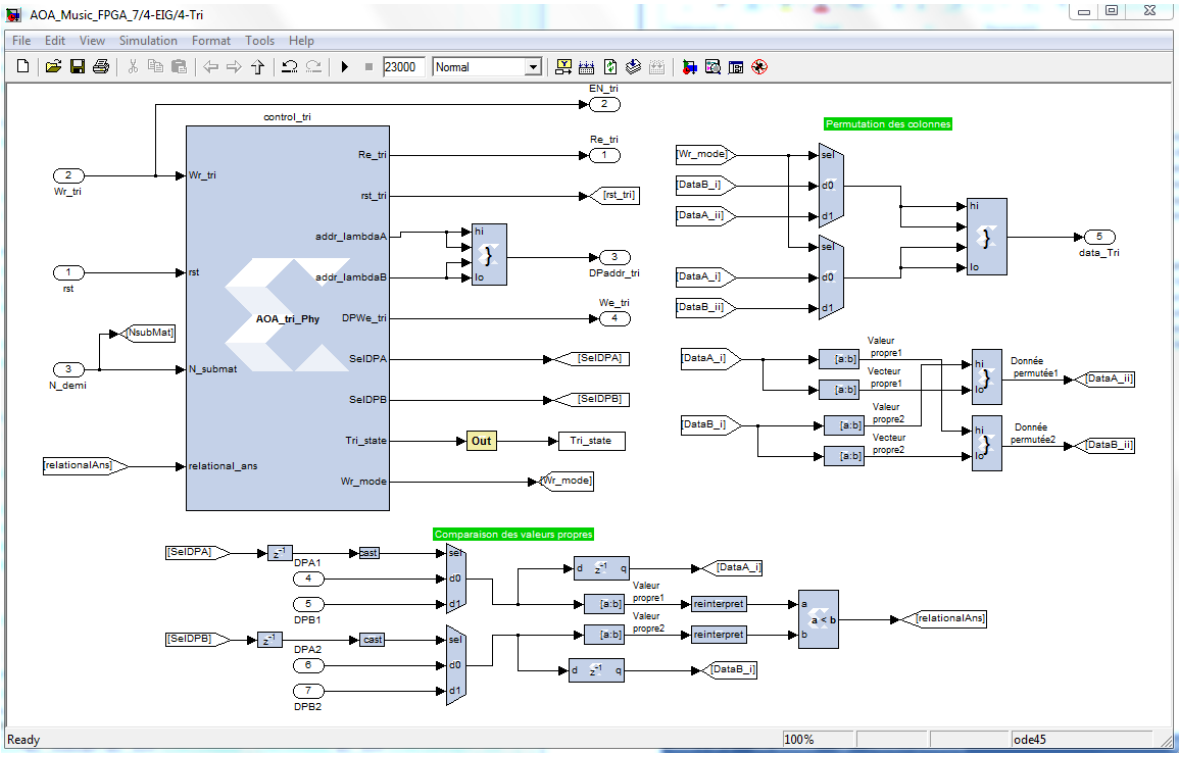


FIGURE 5.18 – Implémentation du *tri* des vecteurs propres sous *Xilinx System generator*

5.6.2 Bloc de détection du nombre de sources

Pour débuter, la détection du nombre de sources, toutes les valeurs propres sont lues depuis les DPs A et B en série de deux à chaque coup d'horloge. Les sorties des deux premiers multiplexeurs sont ensuite additionnées et transmises à un accumulateur via un multiplexeur. Une fois les valeurs propres accumulées, le résultat est envoyé à un registre qui retiendra celle-ci tant que son entrée *Enable* reste à zéro. Ainsi, la somme totale des valeurs propres est fixée à la sortie du registre. Le contenu de ce dernier est une valeur empirique multipliée de 0.95, avant d'être transmis à la première entrée du comparateur.

Ensuite, l'accumulateur est remis à zéro et son entrée est reliée directement à la sortie de l'un des deux premiers multiplexeur. La première valeur propre est lue et accumulée à zéro. Tant que la sortie de l'accumulateur est inférieure au 95% de la somme totale, le compteur est incrémenté et la lecture de la valeur propre suivante est effectuée. Sinon, l'opération est terminée et la sortie du compteur équivaut au nombre de sources.

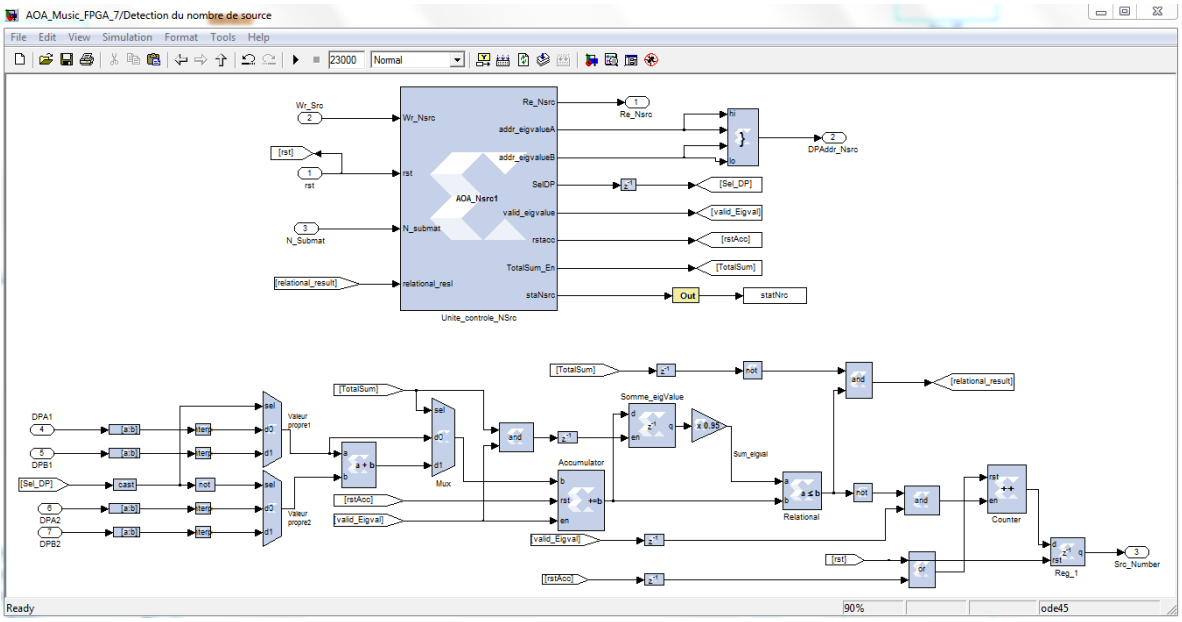


FIGURE 5.19 – Implémentation de la détection du nombre de sources sous *Xilinx System generator*

5.7 Implémentation de la calibration par matrice G

L'implémentation de la calibration par la matrice **G** se base sur l'équation (4.25) et requiert le calcul du vecteur de balayage idéal, puis la matrice **G**, et enfin la calibration de la matrice de covariance complexe. Pour des raisons de clarté, chacune de ces trois opérations a été développée par des sous-systèmes distincts montrés à la figure 5.20.

Le déroulement est le suivant : premièrement, le sous-système *Matrice* **G** déclenche le calcul du vecteur **a** selon (2.7) via sa sortie *Am_start* pour un angle de référence identifié par le paramètre *Theta_Cal* fourni par l'usager. Ensuite, le sous-système *AM* après calcul du vecteur **a**, le transmet au sous-système *Matrice* **G**, déclenchant le calcul de la matrice **G**. Enfin, cette dernière est utilisée par le sous-système *Calibration*, calibrant la matrice de covariance.

5.7.1 Calcul des vecteurs de balayage

Un sous-système est dédié au calcul des vecteurs de balayage **a** nécessaires aussi bien à la détermination de la matrice de **G** qu'au calcul du pseudo spectre. La détermination de ces vecteurs repose sur les équations (2.7) et (2.8) . Cependant, pour des raisons pratiques liées à l'implémentation sur FPGA, la forme exponentielle de l'équation (2.7) est remplacée par la forme cosinus, sinus en s'appuyant sur la formule d'Euler.

Comme le montre la figure 5.21, le schéma bloc de l'implémentation requiert, en plus des blocs arithmétiques, multiplexeurs et compteurs, un bloc *Cordic*. Ce dernier fonctionne en mode cos et sin et dans cette configuration, présente deux entrées : validation de type bit et une phase en radian acceptant le format 2Q*n*. Il a aussi 4 sorties : *tdata_real* pour le cosinus, *tdata_imag* pour le sinus qui sont

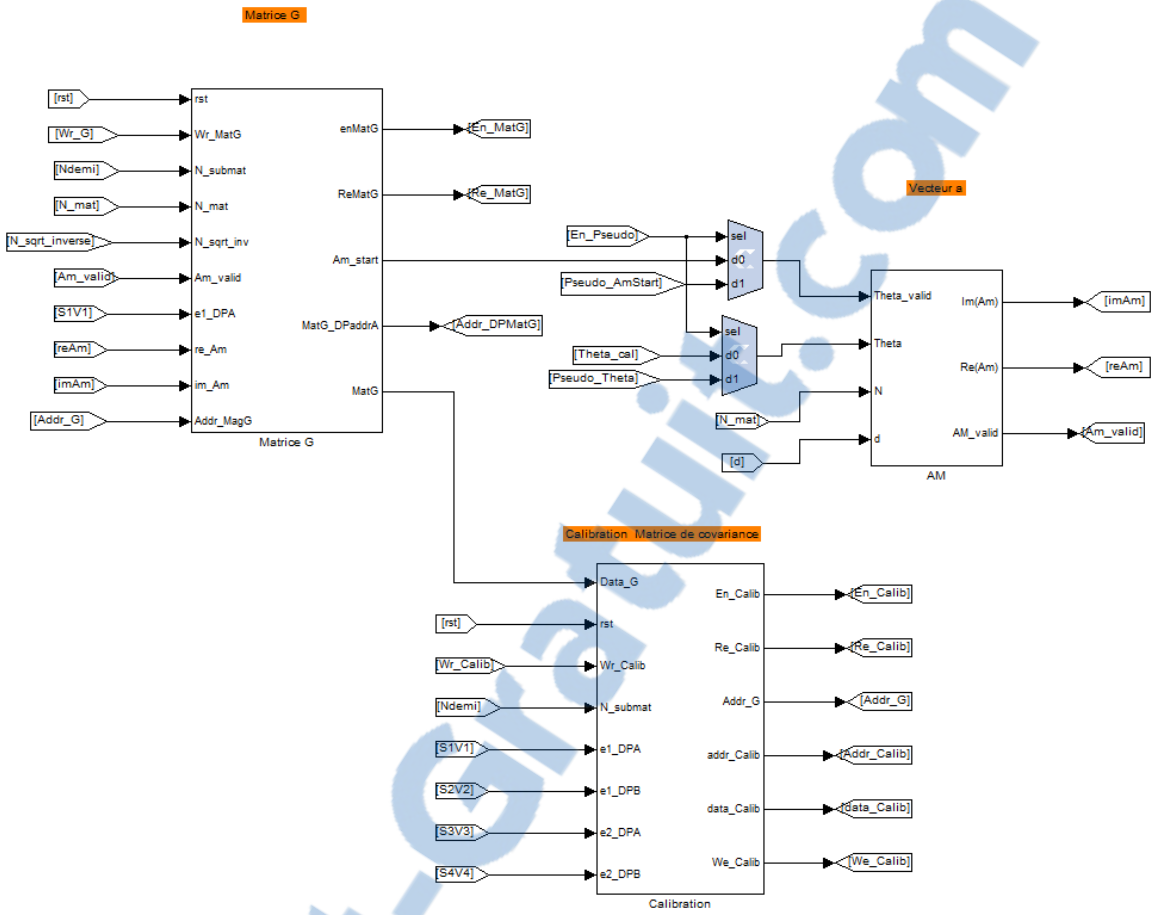


FIGURE 5.20 – Sous-systèmes impliqués dans la calibration de la matrice de covariance sous *XSG*

formatées 1Qn ainsi qu'un bit de validation des signaux de sortie et un bit informant la disponibilité du *Cordic*. L'option *Scale Radian* étant activée, le *Cordic* multiplie toute phase à son entrée par π et donc la phase à l'entrée du *Cordic* devra prendre cela en compte.

L'opération débute lorsque l'entrée du sous-système *Theta_valid* passe de 1 à 0, la structure commence par calculer $\phi(\theta)$ de (2.8). Le multiplexeur identifié *Mux* relie alors sa sortie à son entrée *d1* afin de transmettre l'angle θ multiplié par $1/180$ à l'entrée du *Cordic*. Ce dernier est configuré de sorte que le paramètre à son entrée doit être compris entre -1 et 1. Par conséquent, les paramètres sont ajustés avant d'être transmis au *Cordic*.

À la fin du traitement du *Cordic*, la sortie réelle représentant ϕ est sauvegardée dans un registre. La valeur de ϕ est ensuite multipliée par le paramètre d, la distance entre les éléments du réseau puis multiplié par une série de nombres allant de 0 à $N-1$ fourni par le *compteur1*. Le nouveau paramètre ainsi obtenu est transmis au *Cordic* par le circuit de formatage permettant le respect des contraintes à l'entrée du *Cordic*. Finalement, un circuit aux sorties réelles et imaginaires du *Cordic* retrouve la valeur exacte du sin et du cos du paramètre modifié par les formules trigonométriques. Par exemple, si

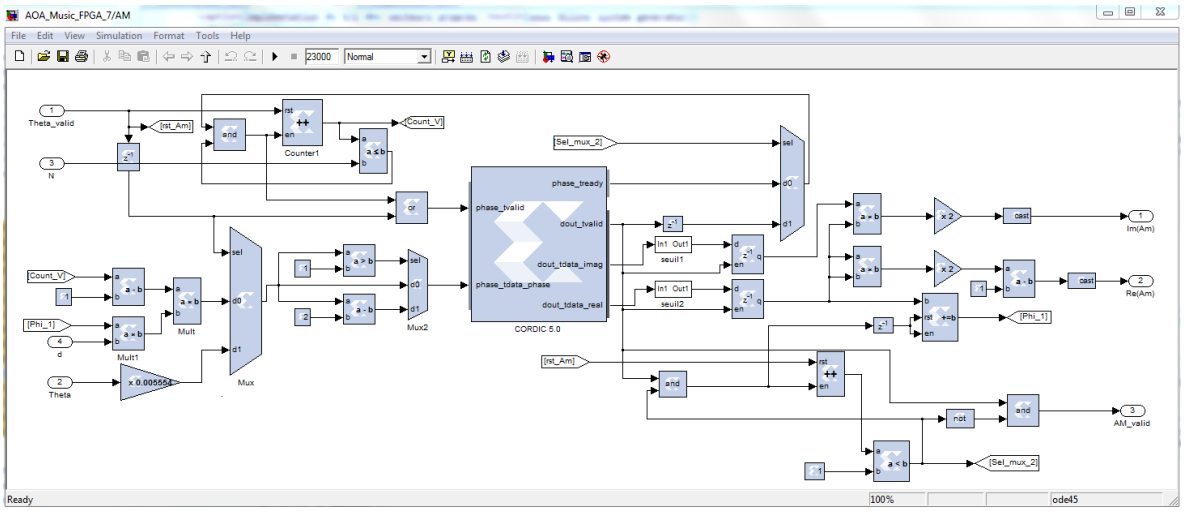


FIGURE 5.21 – Implémetation du calcul du vecteur **a** sous *XSG*

le paramètre original était $2a$ avec $2a > 1$, le circuit de formatage transmettra a au *Cordic*. Par la suite, le $\sin(2a)$ et $\cos(2a)$ souhaités seront retrouvés grâce aux résultats $\sin(a)$ et $\cos(a)$ du *Cordic*.

5.7.2 Formation de la matrice G

La figure 5.22 montre la structure interne réalisée pour le calcul de la diagonale de la matrice **G**. Tout d'abord, l'unité de contrôle déclenche l'opération de calcul du vecteur de balayage **a** et met le système en attente. Une fois le vecteur **a** disponible, l'unité de contrôle procède à la lecture des éléments du vecteur propre de la source de référence. Aussi, les éléments du vecteur **a** sont transmis en série et synchronisés avec les éléments du vecteur propre source lu. Les calculs du numérateur et du dénominateur de l'équation (4.25) sont calculés pour chaque élément et sauvegardés dans un registre à décalage. Ce dernier est utilisé pour réguler le flux des données aux entrées du diviseur. Enfin, à chaque fois que le bit de validation des signaux de sortie du diviseur est actif, le résultat correspondant à un élément de la diagonale de la matrice **G** est sauvegardé dans une RAM interne accessible au sous-système de calibration.

5.7.3 Calibration de la matrice de covariance

La diagonale de la matrice **G** étant générée, il suffit maintenant de l'utiliser pour multiplier à gauche et à droite la matrice de covariance complexe suivant l'équation (4.27). Considérant que la diagonale de la matrice **G** est confinée dans un vecteur **g**, la calibration d'un élément se trouvant à la ligne i et à la colonne j de la matrice de covariance revient à multiplier celui-ci par la résultante d'une multiplication entre l'élément situé à la position i et la conjuguée de la valeur à la position j du vecteur **g**. L'architecture mise en place calibre en parallèle deux éléments à la ligne i et deux autres à $i-1$ de la matrice de covariance.

La figure 5.23 illustre la structure interne du sous-système *calibration*, les encadrés discontinus

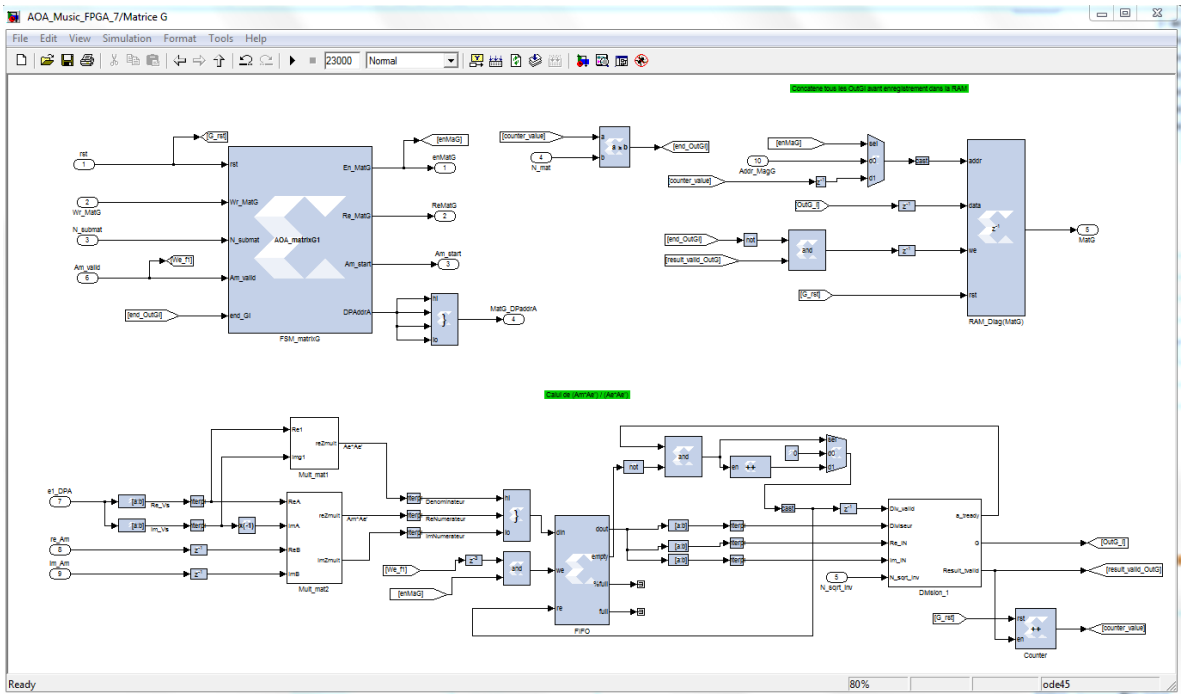


FIGURE 5.22 – Implémetation du calcul de la diagonale de la matrice **G** sous *XSG*

numérotés représentent les principales opérations effectuées. À l'encadré 2 des registres retardés d'un coup d'horloge sont utilisés pour mettre en parallèle quatre éléments du vecteur **g**. Ces éléments sont ensuite scindés en parties réelles et imaginaires et deux d'entre eux sont conjugués avant d'être transmis aux blocs de l'encadré 3. En parallèle, 4 éléments de la matrice de covariance sont lus ; les parties réelles et imaginaires sont extraites avant d'être transmises aux blocs de l'encadré 3. Ce dernier effectue les multiplications et ainsi calibre les 4 éléments de la matrice de covariance. Enfin, les valeurs calibrées sont transmises aux DPs A et B après être concaténées.

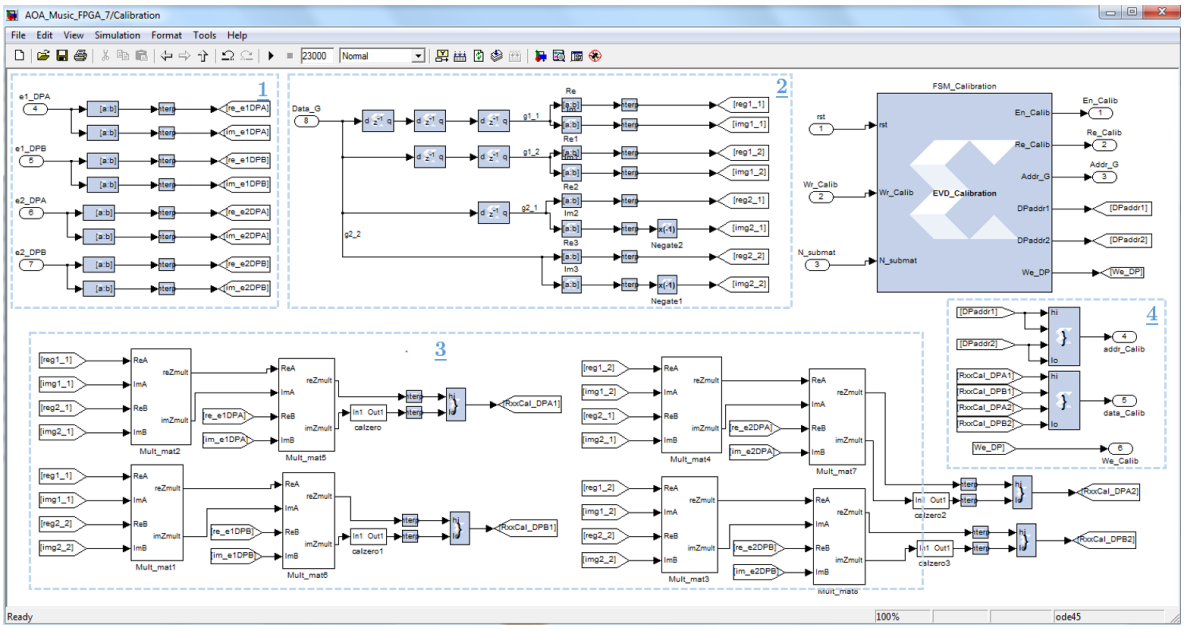


FIGURE 5.23 – Calibration de la matrice de covariance complexe sous *XSG*

5.8 Implémentation du pseudo-spectre MUSIC

Expliqué au chapitre précédent, le pseudo-spectre est défini par le tracé selon les angles d'arrivée résultantes de la projection des vecteurs $\mathbf{a}(\theta)$ dans le sous-espace bruit. Cette projection est caractérisée par le produit scalaire entre les vecteurs $\mathbf{a}$ et les vecteurs propres du sous-espace bruit. La figure 5.24 illustre le schéma bloc de l'implémentation FPGA de l'équation (4.22) pour le calcul du pseudo-spectre.

Toutes les multiplications matricielles réalisées partagent une même architecture série comprenant un bloc de multiplications complexes et deux accumulateurs. Les *accumulateurs 1 et 2* accumulent respectivement les résultats réels et imaginaires des multiplications partielles successives entre les éléments ligne et colonne des matrices opérandes de la multiplication. Le processus débute par le calcul de $\mathbf{P}_n$ exprimé à l'équation (4.23). Pour ce faire les éléments de la matrice $\mathbf{V}_s$ formée par les vecteurs propres sources sont lus depuis les DPs A et B de façon à ce que les entrées Vs_i véhiculent les éléments ligne et Vs_j les éléments colonne. La partie réelle de chaque résultat est augmentée de 1 après une inversion de son signe [1], puis concaténée à la partie imaginaire de signe inverse avant d'être enregistrée dans la mémoire RAM identifiée par $\mathbf{P}_n$ sur la figure 5.24.

La seconde étape du processus est le calcul de la diagonale de la matrice résultante d'une succession de multiplications par des vecteurs allant de $\mathbf{a}(\theta_{\text{debut}})$ à $\mathbf{a}(\theta_{\text{fin}})$ à gauche et à droite de la matrice $\mathbf{P}_n$. Premièrement le vecteur $\mathbf{a}$ est calculé pour l'angle de début de la plage à balayer. Une fois disponible, le vecteur $\mathbf{a}$ est sauvegardé dans une RAM, sa valeur est conjuguée et une première multiplication est réalisée avec la première colonne de $\mathbf{P}_n$. Ensuite, la valeur obtenue est multipliée par la première valeur du vecteur $\mathbf{a}$ et transmis à l'accumulateur *accu3*. Une seconde multiplication entre $\mathbf{a}$ et la

1. Il faut réaliser $1 - V_s V_s^{\dagger}$.

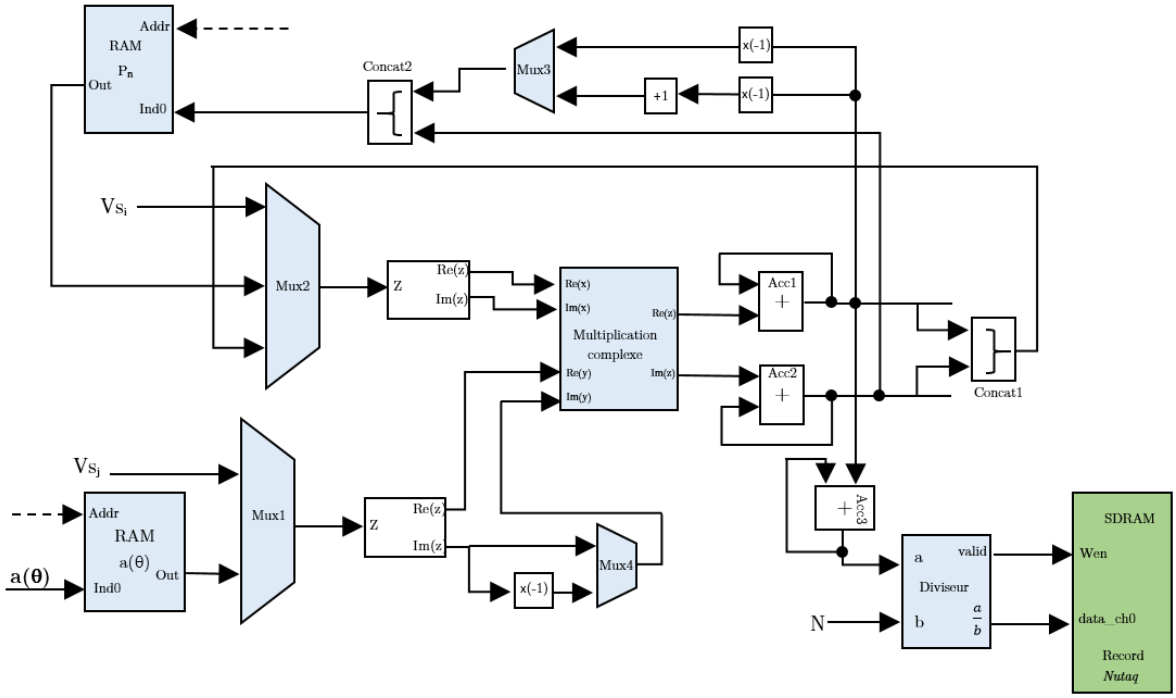


FIGURE 5.24 – Le schéma bloc de l'implémentation du pseudo-spectre de MUSIC

deuxième colonne de $\mathbf{P}_n$ est réalisée, la résultante est multipliée par le second élément du vecteur **a** et accumulée par la précédente valeur de contenu dans *accu3*. Le traitement précédent est réalisé jusqu'à la multiplication avec la dernière colonne de **a**. Ensuite, la valeur contenue dans *accu3* est divisée par *N* et sauvegardée dans la mémoire externe au FPGA SDRAM DDR3 via le module *Record* développé par *Nutaq*. L'opération est répétée à chaque $\theta + 0.5$ jusqu'à la fin de la plage. La résolution adoptée ici sera donc de 0.5°.

5.9 Conclusion

Dans ce chapitre, il a été question de présenter l'implémentation, basée sur l'approche MDL, de l'algorithme MUSIC dans un FPGA afin de déterminer les angles des sources incidentes au réseau d'antennes linéaire. Pour ce faire, plusieurs modules ont été réalisés. Tout d'abord, le module d'acquisition qui regroupe 3 sous-modules à savoir : le filtrage numérique, la normalisation, et le calcul de la matrice de covariance complexe. Les deux premiers sous-modules cités, affine le signal désiré en éliminant un signal parasite ajouté par la carte électronique effectuant la modulation, puis normalise celui-ci. Le troisième sous-module calcule la matrice de covariance complexe des signaux filtrés et normalisés.

Ensuite, dans un souci de réduction de la charge de calcul et le nombre de ressources d'utilisation du FPGA, le module de la transformée unitaire a été réalisé. Ce module à une double fonction : transformer la matrice de covariance complexe en matrice réelle avant la décomposition EVD et effectuer la transformation matricielle inverse du réel au complexe, des vecteurs propres après la

décomposition EVD.

D'ailleurs, la décomposition EVD est le plus gros module réalisé dans ce projet, il implémente le calcul entièrement en réel des vecteurs et valeurs propres. Son modèle a été inspiré de la méthode de Brent [12] et [13] reposant sur l'algorithme de Jacobi. Ce module est en fait un ensemble de 3 sous-modules réalisés et effectuant de façon cyclique les 3 opérations menant à l'obtention des vecteurs et valeurs propres : le calcul des angles θ, la rotation simple et double et l'ordonnancement. Aussi, le bloc *Cordic* de *XSG* a été l'élément clé à l'implémentation des deux premiers sous-modules. Le modèle EVD implémenté est invariant à la taille de la matrice de covariance à son entrée. Ainsi, il peut être utilisé pour toutes applications désirant le calcul des valeurs et vecteurs propres réels. Puisque le module EVD n'assure pas le classement en ordre décroissant des valeurs propres et leurs vecteurs propres associés, un module tri a été développé afin de réaliser cette fonction. La méthode utilisée est celle du tri à bulles.

En outre, l'algorithme MUSIC nécessitant la connaissance du nombre de sources, le module *détection de source* a été implémenté à cet effet. Il est basé sur le critère de l'erreur quadratique moyen RMSE afin de déterminer les valeurs propres les plus significatives. Le module pseudo-spectre MUSIC implémenté réalise la projection des vecteurs de balayage $\mathbf{a}(\theta)$ dans le sous-espace source. Le résultat du module du pseudo-spectre à savoir les échantillons du graphe en fonction des angles θ est enregistré dans la mémoire SDRAM DDR3 reliée au FPGA.

D'autre part, le pseudo-spectre MUSIC donnerait des résultats inutiles et erronés sans le module de calibration élaboré. Ce module implémente la calibration par matrice $\mathbf{G}$ permettant de compenser certaines erreurs rencontrées dans la pratique telles que le déphasage, le déséquilibre I/Q entre les signaux incidents sur le réseau d'antennes.

Enfin, du point de vue de la précision, sachant que le système a été codé entièrement en point fixe, plusieurs réglages ont été effectués tout au long de l'implémentation, afin que les résultats expérimentaux ressemblent à ceux générés à virgule flottante. Le prochain chapitre présente justement les résultats expérimentaux du système implémenté sur FPGA. Il effectue une comparaison entre ces résultats et leur équivalent théorique à virgule flottante.

Chapitre 6

Présentation des résultats obtenus

6.1 Introduction

Ce chapitre met en avant-plan les résultats expérimentaux obtenus suite à l'exécution de MUSIC sur FPGA. Il les compare à ceux provenant d'une simulation à l'aide de code *Matlab* en vue d'analyser la performance du code implémenté.

La première section du chapitre énonce la méthodologie, ainsi que les dispositions à adopter pour une bonne utilisation du produit final. De plus, cette section définit l'interface utilisateur servant à paramétrer le FPGA et visualiser ses résultats.

Avant de mettre en évidence le pseudo-spectre expérimental de MUSIC, la seconde partie du chapitre présente les résultats obtenus à chaque étape du traitement sur FPGA tels que la matrice de covariance complexe, la transformée unitaire et inverse, la décomposition en valeurs et vecteurs propres, en les comparant à leurs équivalents théoriques obtenus par simulation via *Matlab*. Puis l'évaluation de l'angle d'arrivée en présence d'une seule source puis de deux sources corrélées et non corrélées sera commentée. Enfin, un tableau résumant l'allocation des ressources du FPGA au programme implémenté termine le chapitre.

6.2 La méthodologie des tests

Avant tout, il est nécessaire de préciser qu'avant d'obtenir des résultats expérimentaux, il a fallu procéder à une multitude de vérifications des signaux aux différentes étapes du processus, d'acquisitions et d'ajustements divers, tant mécaniques, électriques que logiciels. Autant de problèmes dont la mention sera omise au profit des résultats intéressants.

Le montage de test utilisé en réception et à l'émission a été introduit au chapitre 3. Plus de précisions sur l'ordre de branchement des différents équipements, l'installation des logiciels et actions en rapport avec le FPGA sont apportées à l'annexe A. On suppose dans cette section que le branchement est réalisé et fonctionnel.

D'abord, il faut s'assurer qu'une ou des sources émettent à une distance suffisante de l'antenne réseau pour respecter l'hypothèse du front d'onde plan. Il faut s'assurer aussi que le niveau de puissance est adéquat et, dans le cas de plusieurs sources, que le niveau de corrélation entre elles est bien celui désiré. La variation de la corrélation peut être effectuée par la modulation d'une des sources avec un autre signal. Ensuite, il faut renseigner les paramètres du code implémenté, comme le nombre d'éléments du réseau, le nombre d'échantillons à considérer, et autres données apparaissant sur la figure 6.1, à travers une interface utilisateur de l'environnement *Simulink* exécuté depuis un PC. Cette interface est également utilisée pour le déclenchement du traitement FPGA.

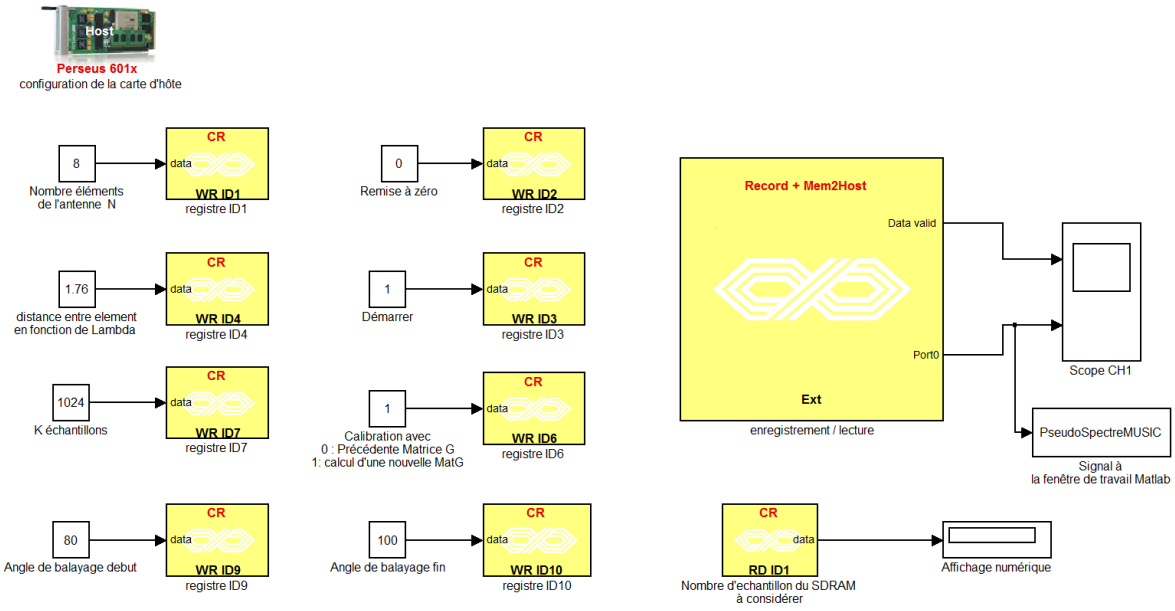


FIGURE 6.1 – Interface utilisateur pour la configuration des paramètres et affichage des données du FPGA avec l'environnement de développement *Simulink*

Le test a débuté par la calibration, avec une source placée à 90° émettant un signal sinusoïdal à 10 GHz. Ce signal émis, propagé dans l'air atteint le réseau d'antennes, puis traverse l'étage RF ayant pour but de ramener le signal en bande de base ou à tout le moins abaisser suffisamment sa fréquence pour qu'il soit échantillonné par la carte FMC. Cette dernière est embarquée dans la *PicoDigitizer* de la carte mère *Persus* sur laquelle est monté le FPGA. Toute la chaine de traitement de la calibration est automatisée, suivie de celle de l'algorithme MUSIC traitée par FPGA. À la fin du traitement, le pseudo-spectre obtenu est enregistré dans la SDRAM. Les données enregistrées dans la mémoire SDRAM sont par la suite lues via l'interface Giga Ethernet vers un PC.

Après s'être assuré du bon fonctionnement de la calibration, les tests ont porté sur la détermination de la position d'une source émettrice déplacée sur une plage d'angles située entre 80° et 100° pour jauger la performance du programme dans la zone sans ambiguïté de notre réseau d'antennes [1] :

$$\begin{array}{rcl} \psi_{s-amb_{\substack{min \\ max}}} & \overset{>}{<} & \arccos\left(\overset{+}{-}\dfrac{2\pi-\beta d}{\beta d}\right) \\ 77^\circ & < \psi_{s-amb} < & 102^\circ. \end{array} \tag{6.1}$$

Enfin, différentes évaluations avec deux sources émettant à la même fréquence soit 10.8 GHz, en faisant varier l'espacement entre elles, ont eu lieu. Afin d'évaluer le pouvoir résolutif du programme, les deux sources ont été séparées d'abord de 4°, émettant à 88 et à 92°. Cet espacement est sensiblement égal à la limite de la résolution (6.2), c'est-à-dire la limite de l'espacement à partir duquel les deux sources sont distinguables par notre réseau d'antennes. Elle est déterminée théoriquement par le critère de Rayleigh, lequel est basé sur l'écart angulaire entre le maximum principal et le premier zéro rencontré sur le diagramme de rayonnement du réseau d'antennes. On obtient [1] :

$$\theta_R = \frac{180}{\pi}\frac{\lambda}{Nd} = \frac{180}{\pi}\frac{\lambda}{8(1.76\lambda)} = 4.07^\circ \tag{6.2}$$

6.3 Analyse et comparaisons entre résultats expérimentaux du FPGA et simulés

6.3.1 Test avec une source émettant un signal dans la zone de non ambiguïté

Grâce à l'outil *ChipScope* de *Xilinx*, il a été possible d'imprimer certains signaux à chaque étape du déroulement de la chaine de traitement du FPGA. Des signaux résultants du calcul de la matrice de covariance complexe, de sa transformée en matrice réelle, la matrice **G** et du vecteur **a**. Des résultats partiels qui s'ils sont erronés sont capables d'engendrer un effet d'avalanche au résultat final. C'est pourquoi il était opportun de comparer chacun d'entre eux avec une valeur de référence supposée exacte, leur équivalent théorique. Ce dernier a été développé en sus et il s'agit d'un script Matlab, émulant le programme implémenté dans le FPGA. Enfin, l'erreur moyenne absolue en pourcentage (*MAPE*) sur 5 essaies entre les résultats expérimentaux et simulés a été calculée suivant la formule de l'équation (6.3). La figure 6.2 dresse le récapitulatif de la moyenne des comparaisons dans le cas d'une source émettant aux positions $86^\circ, 88^\circ, 90^\circ$ et 94°.

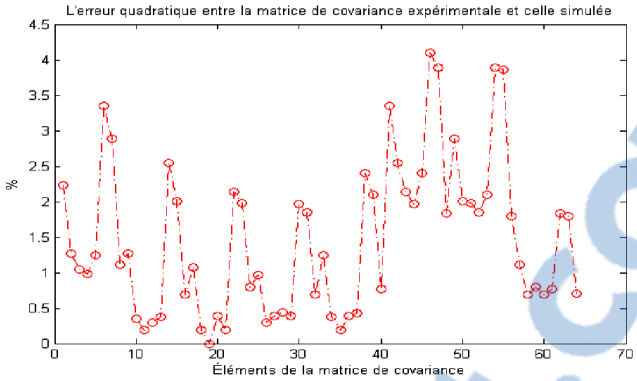
$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|W_{FPGA} - W_{Matlab}|}{|W_{Matlab}|} \tag{6.3}$$

6.3.2 Analyses des résultats

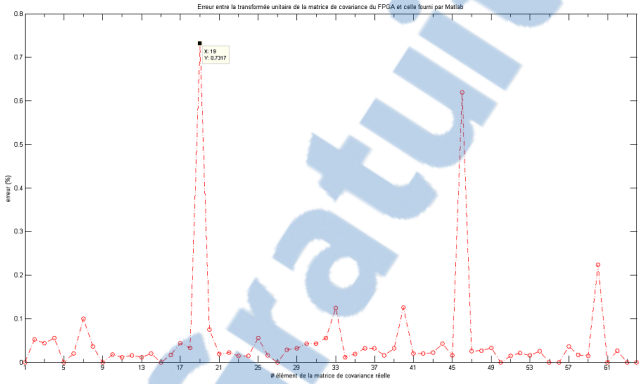
L'architecture à point fixe du FPGA est la première cause non négligeable de la différence de niveaux des résultats observés. En effet, plus le nombre de bits des signaux est élevé, meilleur est le résultat. Les résultats expérimentaux du FPGA sont formatés sur 16 bits signés, alors que les résultats simulés sont à virgule flottante sur 64 bits donc beaucoup plus précis.

Les approximations successives entre les étages de calculs effectués par les blocs arithmétiques de *Xilinx* sont une source de déviation des résultats du FPGA à ceux simulés, notamment la division et la multiplication. Dans le cas de la matrice de covariance, l'omission d'un échantillon sur deux dû au fait du filtrage des 16 canaux par 8 filtres au lieu de 16 utilisés en parallèle pour des raisons de limitations des ressources FPGA, accentue davantage la différence observée.

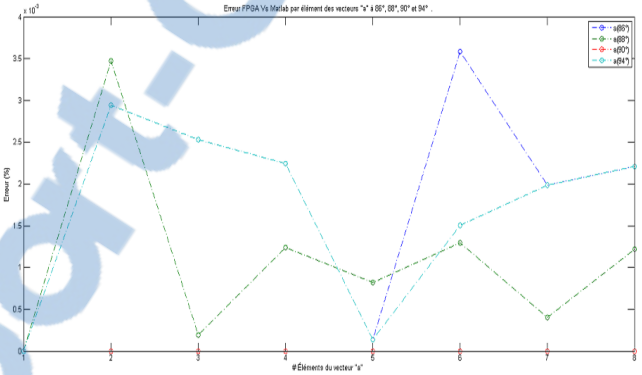
Enfin, l'algorithme *Cordic* utilisé pour calculer le cosinus et le sinus dans le cas du calcul des vecteurs **a**, même performant, reste un algorithme itératif fortement dépendant du nombre de bits de résolution et du nombre d'itérations fournis par l'usager.



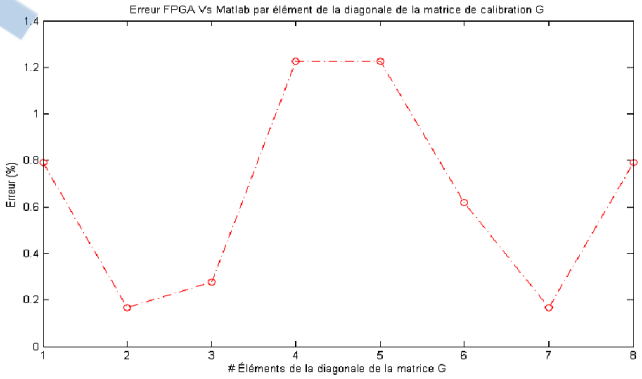
(a) Matrice de covariance complexe



(b) Transformation unitaire de la matrice de covariance complexe



(c) Vecteur **a**



(d) Diagonale de la matrice de calibration **G**

FIGURE 6.2 – Erreurs absolues moyennes des matrices obtenues aux différentes étapes de calcul de MUSIC et de la calibration

6.4 Résultats des valeurs et vecteurs propres expérimentaux et simulés

Les résultats présentés dans cette section portent sur la performance de la décomposition en valeurs et vecteurs propres de la matrice de covariance réelle implémentée. Au total, 5 essais ont été réalisés avec différentes matrices de covariance réelles obtenues avec une source à différentes positions et aussi avec deux sources.

Valeur propre 1	Valeur propre 2	Valeur propre 3	Valeur propre 4	Valeur propre 5	Valeur propre 6	Valeur propre 7	Valeur propre 8
1.5035E-06	1.1266E-6	1.0846E-04	8.2881e-06	1.1229E-04	0.1953	0.1873	0.1318

TABLE 6.1 – Erreur quadratique de chacune des valeurs propres pour une matrice de covariance 8×8

Tout d'abord, les *MAPE* entre les valeurs propres résultantes de la fonction *eig* de *Matlab* et celles fournies par le FPGA sont présentés dans le tableau 6.1. On observe que le niveau d'erreur est élevé dans la zone des valeurs propres les moins significatives par rapport à la zone où l'amplitude des valeurs propres dépasse au moins une unité dans sa partie entière.

Ensuite, les résultats des vecteurs propres sont illustrés dans la figure 6.3. Dans cette figure, il est observé que les courbes labélisées par *data6 data7* et *data8* atteignent jusqu'à 1.8% d'erreur et toutes les autres courbes ont un taux d'erreur pratiquement nulle. Ainsi, la remarque précédente est réitérée, à savoir une distinction de deux zones : une zone (courbe *data1* à *data4*) où les vecteurs propres expérimentaux sont identiques à leur équivalent théorique et l'autre où des erreurs sont observées.

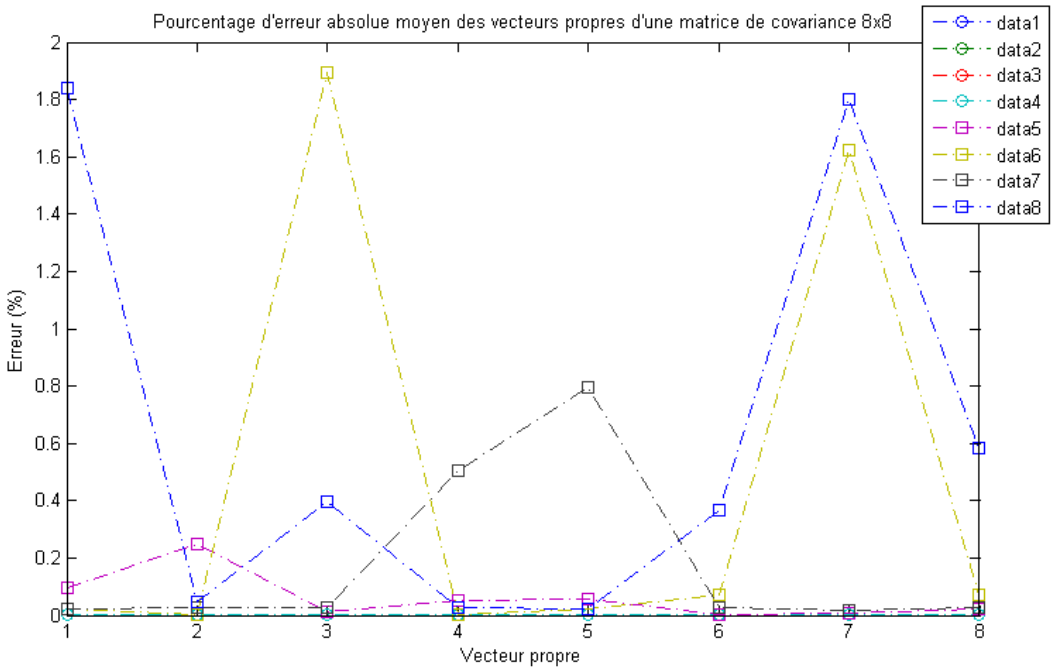


FIGURE 6.3 – Erreur absolue moyenne en pourcentage entre les vecteurs propres obtenus avec FPGA et ceux de *Matlab*

En général, l'erreur est faible dans les deux cas. Heureusement, la zone présentant un niveau d'erreur relativement élevé n'a pas d'influence dans le calcul du pseudo-spectre de MUSIC, car seuls les valeurs et vecteurs propres significatifs, reliés aux sources, sont utilisés.

La principale cause de ces erreurs est imputée au bloc Cordic utilisé pour le calcul de l'angle de rotation. En effet, le *Cordic* en mode *arctan* présente une instabilité dans le traitement avec des nombres quasi nuls en entrée. Ceci provoque des déviations sur les valeurs des angles servant à la détermination des valeurs et vecteurs propres. Pour rappel, le Cordic génère la fonction de sortie $\mathrm{Atan}(\mathrm{Y\_in}/\mathrm{X\_in})$, avec Y_in et X_in données à l'entrée du bloc et Atan, la fonction arc-tangente. Ainsi, le comportement d'instabilité qui peut résulter est lié à la division entre Y_in et X_in, lorsque ceux-ci avoisinent la valeur 0.

6.5 Résultats de pseudo-spectres MUSIC expérimenteaux

6.5.1 Une source

Cette section présente cinq résultats expérimentaux de pseudo-spectres MUSIC obtenus par FPGA avec une source émettant dans la zone de non-ambiguïté puis dans la zone d'ambiguïté. Cette source a été déplacée à 88.5, 90, 100, 102 et 109 degrés. Le même générateur a été utilisé pour produire un signal à la même fréquence pour toutes les positions émettant à 10.8 GHz.

Aussi, le signal de référence situé à 90° a été utilisé pour calibrer le réseau. Les figures 6.4 à 6.7 illustrent les résultats après exécution de MUSIC implémenté sur FPGA.

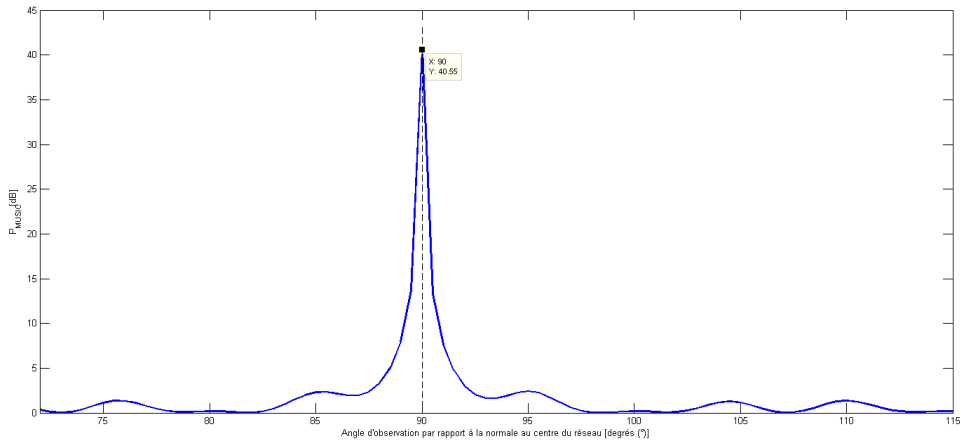


FIGURE 6.4 – Pseudo-spectre obtenu par MUSIC, lorsque la source est à 90° avec données calibrées par matrice de calibration **G** avec une source de référence à 90°.

On remarque à partir des figures que l'angle d'arrivée de la source est bien identifié par le programme implémenté. Il faut mentionner que lorsque la source est positionnée à 90°, le niveau du sommet du pseudo-spectre est plus élevé. Cela est normal puisque le réseau est calibré à cette position. Par contre,

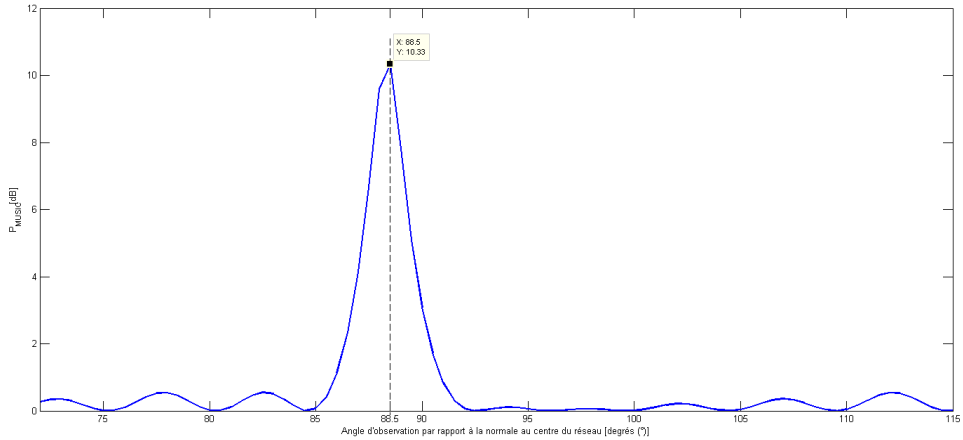


FIGURE 6.5 – Pseudo-spectre obtenu par MUSIC, lorsque la source est à 88.5° avec données calibrées par matrice de calibration **G** avec une source de référence à 90°.

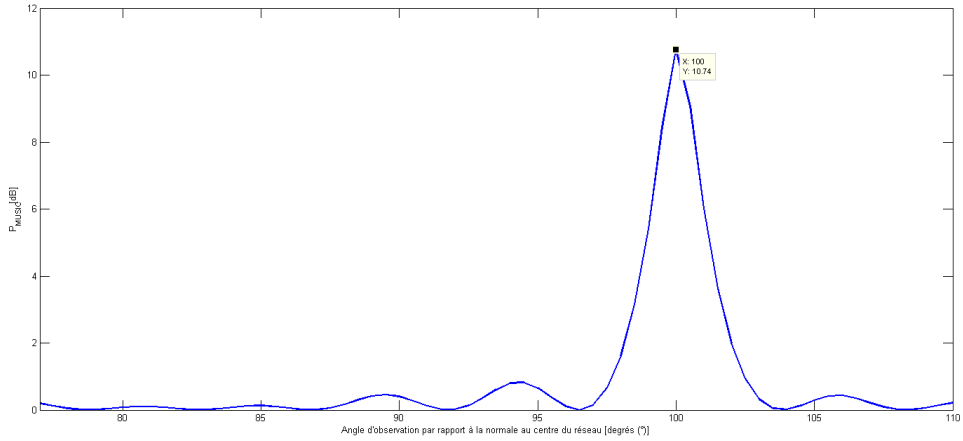


FIGURE 6.6 – Pseudospectre obtenu par MUSIC, lorsque la source est à 100° avec données calibrées par matrice de calibration **G** avec une source de référence à 90°.

un décalage de 0.5° est à souligner lorsque la source est située à 102°. Ce décalage peut être imputé à une erreur de l'observateur, puisque la lecture de l'angle de la source se fait sur un rapporteur gradué à 0.5° d'intervalle, ou à un mauvais alignement du pointeur laser sur le centre du réseau. Une autre cause du décalage observé ici est qu'on atteint la limite de la zone sans ambiguïté. Cette dernière est véritablement observée à la figure 6.8 représentant le pseudo-spectre MUSIC pour une source située à 109°. On voit alors apparaître une autre source près de 76° qui correspond à l'ambiguïté.

6.5.2 Deux sources fortement corrélées

Dans la série de tests avec deux sources, seulement trois graphes du pseudo-spectre MUSIC sont présentés dans ce mémoire. Le premier graphe est montré à la figure 6.9. Il illustre le pseudo-spectre

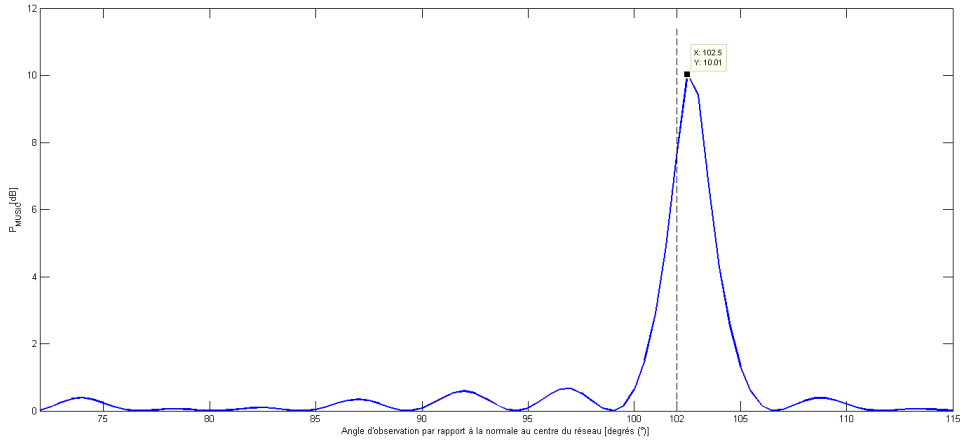


FIGURE 6.7 – Pseudo-spectre obtenu par MUSIC, lorsque la source est à 102° avec données calibrées par matrice de calibration **G** avec une source de référence à 90°.

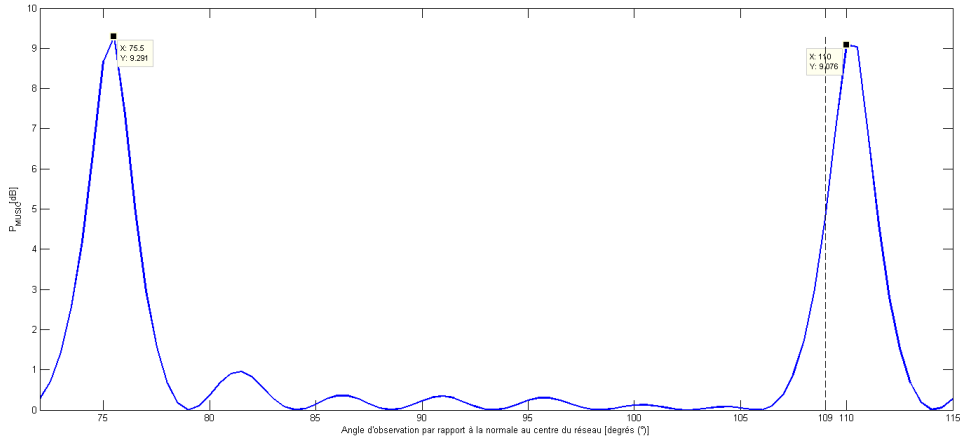


FIGURE 6.8 – Pseudo-spectre MUSIC pour une source située à 109° avec données calibrées par matrice de calibration **G** avec une source de référence à 90°.

MUSIC pour deux sources fortement corrélées en fréquence, d'amplitudes différentes et distancées de 4°, limite de résolution du réseau utilisé. Malgré la forte corrélation et une séparation à la limite du souhaitable, l'exactitude des angles d'arrivée des deux sources est observée par les deux pics cependant bien arrondis sur le graphe du pseudo-spectre. Ceci témoigne du bon pouvoir séparateur du programme implémenté.

La figure 6.10 illustre le deuxième graphe du pseudo-spectre MUSIC dans le cas de deux sources corrélées de même fréquence et d'amplitudes différentes situées à 87° et 93°. Là encore, les angles d'arrivée des deux sources correspondent aux positions des pics observés sur le graphe du pseudo-spectre MUSIC calculé par FPGA. Contrairement au premier graphe, les deux pics sont peu élevés (autour de 4 dB) mais moins arrondis. De plus, il n'existe pas de pics non désirés.

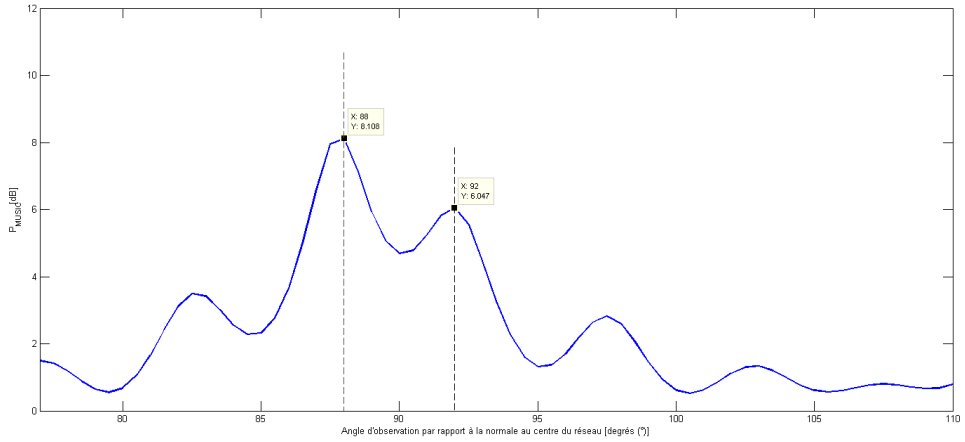


FIGURE 6.9 – Pseudos-pectre MUSIC avec réseau calibré, pour deux sources fortement corrélées situées à 88° et 92°

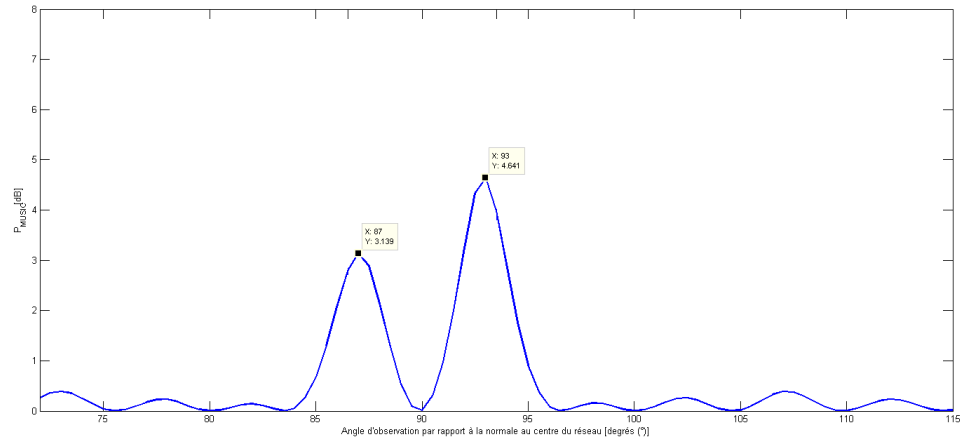


FIGURE 6.10 – Pseudo-spectre MUSIC avec réseau calibré, pour deux sources fortement corrélées situées à 87° et 93°

Le dernier résultat des tests à deux sources est montré à la figure 6.11. Cette dernière désigne le tracé du pseudo-spectre MUSIC, pour des sources situées à 98.5° et à 103.5° corrélées de même fréquence et amplitude. Sur le graphe, on observe deux pics aux angles 98.5° et 104° dont les niveaux sont autour de 8 dB et 4 petites ondulations non désirées, dont le niveau maximum est 3 dB. Une erreur de 0.5° est remarquée à la position de la source située à 2° de plus que la limite supérieure de la zone sans ambiguïté.

Par ailleurs, il aurait été intéressant d'observer ces pseudo-spectres de sources fortement corrélées en appliquant au préalable, un lissage spatial selon l'algorithme décrit à la section 2.3.3. Malheureusement, cela n'a pas été possible, car le temps a manqué pour programmer l'algorithme dans le FPGA.

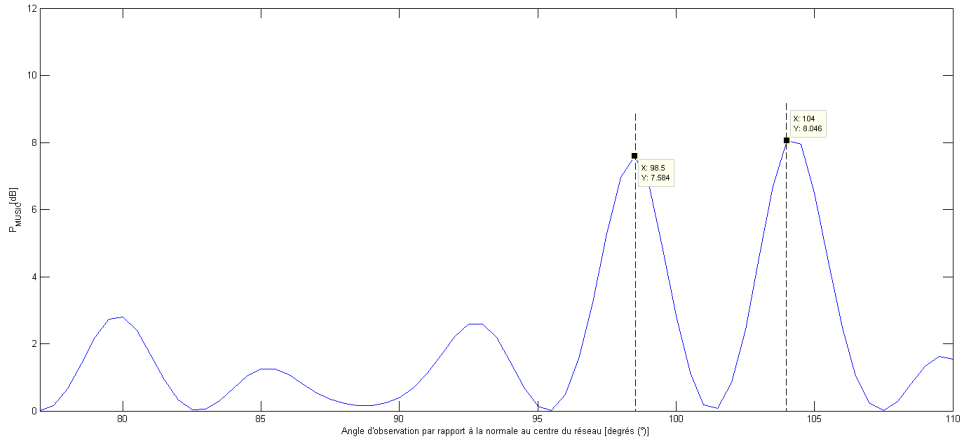


FIGURE 6.11 – Pseudo-spectre MUSIC avec réseau calibré, pour deux sources fortement corrélées situées à 98.5° et 104°

6.6 Temps d'exécution total du traitement de MUSIC par le FPGA

Le calcul du temps d'exécution repose sur la durée de traitement de chaque bloc implémenté et présenté au chapitre précédent. Le tableau 6.2 liste les différents temps d'exécution de chaque bloc par le FPGA. Le détail des calculs est présenté à l'annexe C.

Temps	Description
T_{filtre_norm}	Temps d'exécution du bloc d'implémentation du filtre et de la normalisation. Il est exprimé par C.2.
T_{cov}	Temps d'exécution pour le calcul de la matrice de covariance. Il est exprimé par C.5.
T_{ut}	Temps d'exécution de la transformée unitaire. Il est exprimé par C.7.
T_{ut_inv}	Temps d'exécution de la transformée unitaire inverse. Il est exprimé par C.6.
T_{evd}	Temps d'exécution de la décomposition EVD. Il est exprimé par C.11.
T_{tri}	Temps d'exécution du bloc *tri*. Il est exprimé par C.12.
T_{nsrc}	Temps d'exécution du bloc de détection du nombre de sources. Il est exprimé par C.13.
T_{vecta}	Temps d'exécution du bloc de calcul du vecteur de balayage. Il est exprimé par C.14.
T_{matG}	Temps d'exécution du bloc de calcul de la diagonale de la matrice **G**. Il est exprimé par C.15.
T_{cal}	Temps d'exécution du bloc de calibration. Il est exprimé par C.16.
T_{pseudo_MUSIC}	Temps d'exécution du bloc de calcul du pseudo-spectre de MUSIC. Il est exprimé par C.19.

TABLE 6.2 – Liste des temps de traitement FPGA des blocs implémentés

Sur la base de ces temps d'exécution ci-dessus, on peut maintenant estimer la durée du traitement de l'algorithme de MUSIC par le FPGA. En fait, ce temps dépend du mode de calcul : avec détermination

de la matrice **G** avant la calibration de la matrice de covariance ; ou sans détermination de la matrice **G**, lorsque cette dernière est connue à l'avance.

Dans le cas du traitement avec la détermination de la matrice **G**, le temps d'exécution total du FPGA s'écrit comme suit :

$$\begin{aligned} T_{total\_MUSIC1} &= T_{filtre\_norm} + 2\left(T_{cov} + T_{ut} + T_{evd} + T_{tri}\right) + \\ &\max\left(T_{ut\_inv}\ ,\ T_{vecta}\right) + T_{matG} + T_{cal} + T_{nsrc} + T_{ut\_inv} + T_{pseudo\_MUSIC} \end{aligned} \tag{6.4}$$

où $\max(T_{ut\_inv}, T_{vecta})$ signifie le temps maximal entre le temps de calcul de la transformée unitaire inverse et celui du bloc de calcul du vecteur **a**, car ces deux blocs sont exécutés en parallèle. Alors que dans le second mode, l'équation est exprimée par :

$$T_{total\_MUSIC2} = T_{filtre\_norm} + T_{cov} + T_{cal} + T_{ut} + T_{evd} + T_{tri} + T_{nsrc} + T_{ut\_inv} + T_{pseudo\_MUSIC}. \tag{6.5}$$

Variable	Description
N	nombre d'éléments du réseau d'antennes égal à 8
n	nombre de bits égal à 16
$d1$	nombre d'échatillons retenus pour le calcul des valeurs minimum et maximum égal à 1524
M	nombre de sources égal à 2
H	nombre d'itérations de la décomposition EVD égal à 21
N_{sample}	nombre d'échantillons considérés dans le calcul de la matrice de covariance égal à 256
T_{clk}	Durée d'un coup d'horloge du FPGA. Sachant que la fréquence d'horloge est de 125 MHz, T_{clk} est égal à $\frac{1}{125\,\text{MHz}} = 8\,\text{ns}$

TABLE 6.3 – Valeurs numériques utilisées dans les essais effectués

Les valeurs numériques utilisées dans le cadre de notre projet sont listées dans le tableau 6.3. À partir de ces valeurs, l'estimation des temps d'exécutions du FPGA pour le mode avec détermination de la matrice **G** est :

$$T_{total\_MUSIC1} = 15.383\,\text{ms}$$

La valeur numérique du temps d'exécution total du FPGA dans le second mode est la suivante :

$$T_{total\_MUSIC2} = 13.295\,\text{ms}$$

6.7 Résumé d'utilisation des ressources FPGA

L'allocation des ressources du FPGA (*Virtex-6*), après la synthétisation des blocs implémentant la calibration des données, l'algorithme MUSIC et les filtres numériques, est montrée dans le tableau 6.4.

Type de ressources	Utilisées	Disponibles	% d'utilisation
Nombre de Slices Registres	34552	301440	11%
Nombre de Slices LUT logique	47479	150720	31%
Nombre utilisés comme logique	42333	150720	28%
Nombre utilisés comme memoire	3720	58400	6%
Nombre de Blocs RAM	40	832	4%
Nombre de Slices occupées	15997	37680	42%

TABLE 6.4 – Allocation des ressources FPGA sous un *Virtex-6 LX240T*

Dans ce tableau, il est lu que 15997 des 37680 slices du *Virtex-6 LX240T* sont utilisées, ce qui implique environ 42% de sa capacité. Cette allocation ne dépassant pas les 50% d'occupation des ressources, il reste assez d'espace pour implémenter des algorithmes supplémentaires dont un lissage spatial par exemple.

6.8 Conclusion

La présentation des résultats expérimentaux a été le sujet couvert par ce chapitre. Primo, la méthodologie adoptée lors de la phase des tests a été dressée. Les prémices, les conditions et les manipulations à respecter avant les tests ont été citées. Il a également été mention de l'interface utilisateur permettant la paramétrisation du programme implémenté dans le FPGA.

Secundo, l'analyse des erreurs absolues moyennes entre les résultats expérimentaux et simulés à des étapes importantes de la chaine d'exécution du programme implémenté a été présentée. Les différents graphes d'erreurs illustrés montrent un écart très faible entre les résultats comparés. Cela témoigne de la bonne fonctionnalité des blocs conçus à cet effet. Les résultats de ces derniers à savoir la matrice de covariance, la résultante de la transformée unitaire, le vecteur **a** et la diagonale de la matrice **G** s'ils sont erronés provoquent un effet avalanche sur le pseudo-spectre.

Tertio, la performance du bloc de décomposition EVD a été justifiée par le faible niveau des erreurs MAPE des valeurs et vecteurs propres observés. Mieux, les taux d'erreurs les plus bas ont été observés dans les positions des valeurs et vecteurs propres principaux ayant un impact significatif dans la suite du projet.

Ensuite, les graphes des pseudo-spectres de MUSIC calculés par FPGA en présence d'une source puis de deux sources ont été présentés. Dans le cas des tests en présence d'une seule source, dans la zone de sans-ambiguïté le programme s'est bien comporté. Le pic observé sur les différents graphes coïncidait bel et bien avec l'angle d'arrivée de la source incidente au réseau d'antennes. En outre, il a été noté que le pic était plus affiné lorsque la source était positionnée à 90° (angle de référence ayant servi à

la calibration du réseau d'antennes). D'autre part, lorsque la source a été positionnée aux extrémités immédiates de la zone de non-ambiguïté, une incertitude 0.5° a été observée, ce qui reste bien en deçà de la résolution du réseau qui est de 4°.

Le calcul du temps d'exécution du programme par le FPGA a été effectué. Le temps d'exécution approximatif du programme par le FPGA en considérant la configuration adoptée pour les tests est autour de 15 ms. Ce temps comprend le calcul d'une nouvelle matrice de calibration $\mathbf{G}$ avant le déroulement de la calibration et de MUSIC. Il est considéré satisfaisant au regard des exigences du présent sujet de maitrise. Il serait donc possible de travailler en temps réel avec une estimation des angles d'arrivée à 60 Hz. Enfin, un tableau a été dressé montrant la proportion des ressources du programme implémenté dans le *Virtex-6*.

Chapitre 7

Conclusion générale

7.1 Accomplissements

Au terme de cette maitrise, un système embarqué permettant l'estimation des angles d'arrivée sur un réseau d'antennes a été réalisé. L'objectif principal était l'implémentation de l'algorithme MUSIC en logique FPGA dans l'optique de déterminer en temps réel la ou les directions d'arrivée de sources émettant à 10 GHz des signaux électromagnétiques sur un réseau uniforme et linéaire de 8 antennes. Une calibration pseudo-automatique du réseau d'antennes a été effectuée en vue de remédier à toutes erreurs pouvant perturber le modèle statistique des données acquises, sans quoi une estimation de MUSIC est impossible. Le *PicoDigitizer*, un produit du partenaire à ce projet, *Nutaq*^{TM} a été utilisé faisant office combiné de carte d'acquisition et de logique FPGA. Une évaluation du taux d'erreurs sur les résultats des blocs critiques et du temps d'exécution du système implémenté a été réalisée. En addition, plusieurs expérimentations du système embarqué pour des cas réels ont eu lieu. Les résultats obtenus et présentés au chapitre précédent justifient que le système réalisé est bel et bien fonctionnel.

Challenge de la calibration en temps réel

Parmi les 3 catégories de calibration existant dans la littérature, celle qualifiée d'automatique est celle qui rencontre à des exceptions près les exigences pour les applications en temps réel. Cependant, aucune méthode de cette catégorie recensée au cours des lectures menées dans le cadre du projet ne trouvait de solution aux erreurs critiques engendrées par le réseau d'antennes et la carte électronique de modulation. De plus, les quelques preuves de fonctionnalité de ces techniques de calibration restaient théoriques. Il a nécessité plusieurs épreuves sur des données réelles avec différentes techniques de calibration pour aboutir à celle de la calibration par matrice $\mathbf{G}$. À cette dernière, il a été ajouté une technique issue de notre cru. Ensemble, ces deux méthodes ont permis de résoudre en partie la problématique. Il a fallu remplacer des composants électroniques défectueux sur la carte électronique de modulation et prévoir l'ajout de filtres numérique dans la logique FPGA pour traiter en totalité la problématique de la calibration.

Défis du RCP : *XSG* et *MBDK* avec l'environnement *Simulink*

Le *System generator* de *Xilinx* et le MBDK de *Nutaq* sont de bons outils pour le contrôle de prototypage rapide. En effet, le temps sauvé grâce à la riche bibliothèque de blocs de ces 2 outils pour implémenter le système de haut niveau de cette maitrise a été un réel avantage. Aussi, le fait qu'ils offrent la possibilités d'utiliser l'interface de programmation graphique familier *Simulink* est agréable. Cependant, avec toutes les fonctionnalités, choix et options dans les blocs, la phase d'apprentissage des bibliothèques est assez intense. Par ailleurs, la flexibilité de ces blocs nécessite parfois l'ajout de nouveaux circuits spécifiques pour optimiser leurs résultats. Cela occasionne une augmentation de l'allocation des ressources FPGA.

Défis de l'implémentation de MUSIC en logique FPGA

Comme il fallait s'y attendre, l'implémentation de MUSIC a été le défi majeur de cette maitrise. En effet, l'algorithme MUSIC est une méthode à haute densité de calculs. L'implémenter dans sa forme naturelle dans un FPGA nécessiterait énormément de ressources et de temps de calcul. Il était donc impératif de l'adapter. Ainsi, la première action a porté sur le calcul de la matrice de covariance complexe. Profitant de la caractéristique à symétrie hermitienne de celle-ci, les opérations de multiplications effectuées en parallèle ont porté sur la moitié de la matrice et l'autre moitié a été reconstituée par simple changement de signe. Cette action balance efficacement le temps de calcul, le nombre de ressources et la précision des résultats. Par contre, l'architecture parallèle employée limite son usage à 16 canaux.

Ensuite, au lieu d'effectuer la décomposition EVD de la matrice de covariance directement en complexe comme à l'accoutumée ce qui serait très gourmand en ressource, l'action entreprise a été de la convertir en matrice réelle. Cette manière de procéder a eu pour avantage de sauver un grand nombre de ressources logiques et du temps de calcul par la même occasion. Cependant, la précision des éléments des vecteurs propres a été légèrement affectée.

La troisième intervention s'est orientée sur le choix d'une technique de décomposition EVD à faible densité de calculs avec une bonne précision. La meilleure candidate a été la méthode de Jacobi. Cette méthode itérative a inspiré l'élaboration d'une nouvelle architecture en série, permettant la décomposition en valeurs et vecteurs propres réelle. Cette architecture en plus de fournir des résultats comparables à ceux de la fonction *eig* de Matlab, peut être réutilisée pour les matrices symétriques réelles de toutes tailles. Par contre, elle demande un temps de traitement moyennement élevé comparativement aux architectures en parallèle. En somme, avec une allocation en ressources de la FPGA *Virtex-6* inférieure à 50% et un temps de traitement au tour de 13 ms, les résultats obtenus sont très satisfaisants.

7.2 Travaux futurs

Ce travail de recherche vient combler en partie le manque de travaux expérimentaux à succès dans le domaine du traitement des antennes appliqué aux systèmes embarqués en temps réel. Cependant, malgré les accomplissements, il y a toujours place à l'amélioration et à de possibles extensions. Bien

sûr, cette continuation doit être guidée ou motivée par l'optimisation en espace (ressources logiques), en temps et en performance des blocs et algorithmes.

L'amélioration immédiate qui pourrait venir à l'esprit est l'ajout de la méthode de lissage spatial bidirectionnel présentée au chapitre 1. Cette technique en plus d'améliorer le pouvoir séparateur de MUSIC en présence de sources corrélées, viendrait compenser dans une certaine mesure le manque de précision engendrée par la transformée unitaire.

Une autre éventuelle extension serait de remplacer la technique de calibration par matrice implémentée par une méthode utilisant les réseaux de neurones. Cette proposition de réalisation va entièrement automatiser le processus de calibration. Ce dernier dans sa forme actuelle par matrice **G** requiert une intervention humaine. Celle avec réseau de neurones exigeait beaucoup trop d'intervention, d'où son rejet dans le cadre de ce mémoire. Si la méthode était automatisée cependant, elle permettrait une meilleure calibration qui considérerait aussi le couplage entre les éléments. Il faut une quantité énorme de mesures pour l'apprentissage supervisé du réseau. Entre autres techniques de calibration utilisant les réseaux de neurones, l'exemple qui peut être cité est la méthode ADALINE développée par Hugo Bertrand et *al.*[5].

Enfin, la prochaine étape serait l'amélioration du bloc implémentant le calcul de la matrice de covariance complexe. Tel qu'il est implémenté actuellement, ce bloc est rigide et limite l'utilisation du FPGA à 16 canaux maximum. Certes, modifier le bloc dans l'actuelle architecture parallèle pour fixer un nombre maximal de canaux à chaque fois que cela est nécessaire est simple. Cependant dans une vision pérenne et pragmatique, une architecture plus flexible permettant une sélection à travers une interface de contrôle *Simulink* ou autre de canaux plus grands sans modifications du code serait bénéfique. Ceci, d'autant plus que les récentes recherches portent sur des réseaux comportant plus de 12 antennes, c'est à dire 24 canaux en considérant les branches I et Q. D'autre part, cela contribuera à un gain en ressources logiques FPGA. Une piste et un début de recherche à la mise en place de cette proposition pourraient se trouver dans la thèse de Minseok Kim [8].

Annexe A

Annexe A : Liste du matériel et installation des logiciels

Réf.	Désignation	Utilisation
(6)	Dual DC Power Supply HP 6205B $(0-40\,\mathrm{V})$	Alimentation DC 12 V électronique
(3)	Signal Generator HP 8647A $(200\,\mathrm{kHz}-1000\,\mathrm{MHz})$ $(0-40\,\mathrm{V})$	Source 2^{e} mélangeur 272.88 MHz
(8)	Générateur RF Lab-Volt 9505-01 $(10.53\,\mathrm{GHz})$	Source du 1^{er} mélangeur 10.53 GHz
(11)	Synthesized Generator HP 8673C $(0.05-18.3\,\mathrm{GHz})$	$1^{\mathrm{ère}}$ source émettrice $10.8\,\mathrm{GHz}+10\,\mathrm{dBm}$ à $-10\,\mathrm{dBm}$
(7)	Microwave Amplifier HP 8349B $(2-20\mathrm{GHz})$	Amplificateur pour le 1^{er} mélangeur
(1)	8 Antennes cornet Lab-Volt 9535-00	Antennes qui forment le réseau
(1a)	8 Adaptateurs guide d'ondes-SMA Arra 90-422	Adaptation des antennes du réseau aux premiers câbles
	8 Câbles Balden $4'$ bleus Storm SMA-SMA	Câbles entre l'adapteur et la $1^{\mathrm{ère}}$ étage de mélangeurs
(2a)	8 Mélangeurs ST Microwave Corp. STDB-2001-AM	$1^{\mathrm{ère}}$ étage de mélangeurs
(2f)	Diviseur de puissance 8 voies Triangle Electronics YE-68	Diviseur de puissance pour la source des 8 mélangeurs du 1^{er} étage
	8 Câbles Balden $2.5'$ orange SMA-SMA	Câbles entre le 1^{er} étage de mélangeurs et les cartes électronique
(2b)	8 amplificateurs Mini-Circuits MAN-1LN	Amplificateur des signaux
(2c)	8 diviseurs de puissance 2 voies Mini-Circuits PSC-2-1	Division de la puissance des signaux pour attaquer le 2^{e} étage de mélangeurs
(2e)	8 diviseurs de puissance $0-90^{\circ}$ Mini-Circuits SCPQ-400	Division de la puissance de la source du 2^{e} étage de mélangeurs en phase (I) et en quadrature (Q)

(2g)	2 diviseurs de puissance 8 voies Synergy DSK-716S	Division de la puissance de la source du 2^{e} étage de mélangeurs
(2g)	16 mélangeurs Mini-Circuits TUF-3SM	le 2^{e} étage mélangeurs
	ERF8 breakout to 16 SMA	Câbles reliant le 2e étage de mélangeurs et le *PicoDigitizer*
(9)	1 antenne cornet Scientific Atlanta 12-8.2	Pour la 1^{re} source émettrice et possédant le plus grand gain
(10)	1 antenne cornet Lab-Volt 9550	Pour la 2^{e} source émettrice
	Pointeur laser et rapporteur d'angles	Pour l'évaluation visuelle de la position des sources
(5)	Spectrum Analyzer HP 8653E (9 kHz – 26.5 GHz)	Pour la vérification des spectres des signaux à la réception et à l'émission avant le test avec le FPGA
(4)	PicoDigitizer 125-Series/16, non-embedded version equiped with : Perseus601x Carrier board with a Virtex6 FPGA (LX240T) ; 4 GB SODIMM DDR3 ; MI125 FMC (16 ADC channels)	Pour l'acquisition des 16 canaux et le traitement de l'algorithme MUSIC
	PC Toshiba Core 2 Duo 2.53 GHz, 4 GB RAM	Pour la configuration du FPGA, le développement et l'affichage des résultats

TABLE A.1 – Liste du matériel utilisé dans le cadre du projet

Annexe B

Annexe B : Exemple de résultats expérimentaux

Dans cette annexe, un exemple de résultats obtenus durant la phase d'expérimentation avec deux sources est présenté. Le tableau B.1 compare les valeurs et vecteurs propres calculés par FPGA et ceux obtenus par la fonction *eig* de Matlab. La dernière ligne du tableau est le résultat du produit matriciel $\mathbf{V}_{Matlab}\mathbf{V}_{FPGA}^{\dagger}$ en vue de vérifier la colinéarité entre les vecteurs propres expérimentaux et théoriques. Ainsi, ce produit est la preuve que les deux premiers vecteurs propres (ceux sources) sont bien orthogonaux entre eux et avec les vecteurs propres bruit.

Matlab : Valeurs propres

5.2313	0	0	0	0	0	0	0
0	4.0361	0	0	0	0	0	0
0	0	0.0355	0	0	0	0	0
0	0	0	0.0148	0	0	0	0
0	0	0	0	0.0020	0	0	0
0	0	0	0	0	0.0012	0	0
0	0	0	0	0	0	0.0010	0
0	0	0	0	0	0	0	0.0007

FPGA : Valeurs propres

5.2417	0	0	0	0	0	0	0
0	4.0808	0	0	0	0	0	0
0	0	0.0361	0.0010	0	0	0.0012	0.0010
-0.0012	0	0	0.0173	0	0	0	0
0	0	0	0	0	0	0	0
0.0012	0	0	0	0	0	0	0
-0.0012	0	0	0	0	0	0	0
0	0.0012	0	0	0	0	0	0

Matlab : Vecteurs propres

0.6358	0.6103	0.0533	0.0078	0.0126	-0.3998	-0.1553	-0.0394
0.4704	-0.1840	-0.1365	-0.1367	-0.4459	0.0613	0.5556	-0.5877
0.4998	-0.0593	0.0095	0.0962	0.3383	0.7262	-0.1542	0.5565
0.3532	-0.7682	0.0601	0.0367	0.0882	-0.3875	-0.2472	0.0686
0.0008	-0.0007	0.1724	0.6286	-0.1355	-0.1621	0.6322	0.0600
0.0003	0.0022	-0.4732	-0.1226	0.7050	-0.3481	0.2594	-0.2892
-0.0023	0.0062	-0.5649	-0.3406	-0.3884	0.0801	0.1523	0.4956
0.0047	-0.0102	0.6341	-0.6667	0.1123	-0.0704	0.3035	0.0766

FPGA : Vecteurs propres

0.6355	-0.6096	0.0500	0.0271	-0.0078	0.0171	0	-0.4695
0.4705	0.1848	-0.1355	0.0708	-0.3350	0.6602	-0.0073	0.4160
0.5002	0.0596	0.0176	-0.1082	0.2849	-0.5728	0.0049	-0.5728
0.3525	0.7686	0.0569	0.0073	0.0510	-0.1033	0.0063	-0.5183
0	0	0.1953	-0.6230	0.5750	0.4053	0.2808	-0.0073
0	-0.0020	-0.4761	0.1541	0.5288	0.1736	-0.6619	-0.0422
-0.0022	-0.0061	-0.5654	0.3684	0.2485	0.0044	0.6943	-0.0369
0.0044	0.0100	0.6260	0.6597	0.3635	0.1755	0.0334	0.0959

Colinéarité des vecteurs propres FPGA et Matlab

0.9999	0.0007	0.0012	0.0008	-0.0004	-0.0001	-0.0002	-0.5717
0.0008	-1.0002	-0.0004	0.0007	-0.0006	0.0006	-0.0015	0.0678
-0.0004	-0.0002	1.0000	0.0211	-0.0099	-0.0043	-0.0080	-0.0180
-0.0005	-0.0004	-0.0209	-0.9954	0.0446	-0.0340	0.0006	-0.1854
-0.0001	-0.0000	-0.0002	0.0569	0.4894	-0.4116	-0.7651	-0.4346
0.0009	-0.0000	0.0071	-0.0575	-0.1133	-0.4804	0.2388	0.0042
-0.0005	-0.0002	0.0053	-0.0462	0.4081	0.8336	0.1154	0.5286
-0.0481	-0.1058	0.0007	0.0478	0.3848	-0.7106	0.5615	-0.5084

1

2

TABLE B.1 – Résultat de la décomposition en valeurs et vecteurs propres en présence de deux sources

Annexe C

Temps d'exécution du programme

C.1 Annexe C : Temps d'exécution du bloc d'implémentation du filtre et de la normalisation

Le temps consommé pour filtrer et normaliser les signaux I et Q, est la somme de T_{filtre} pour le temps d'exécution du bloc filtre et T_{norm} pour le temps d'exécution de la normalisation. En ce qui concerne T_{filtre}, il est équivalent au temps de latence initial du bloc *FIR compiler 6.3* de *XSG* additionné au temps de latence d'un registre. Le temps de latence initial du bloc *FIR* de *XSG*, selon [18] équivaut au nombre de coefficients du filtre, qui est aussi égal au nombre de bits d'un échantillon, divisé par 2, le tout soustrait de 1 et multiplié par le temps d'un cycle d'horloge T_{clk}. Soit l'équation ci-après

$$T_{filtre} = \left(\frac{n}{2} - 1\right) T_{clk} \tag{C.1}$$

où n est égal au nombre de bit et T_{clk} est le temps d'un cycle d'horloge. Au chapitre 5.2.2, le bloc de normalisation est en fait représenté comme étant une succession d'opérations impliquant 3 sous-blocs : *min-max*, *ajustement* et *diviseur* ; reliés les uns après les autres. Premièrement, il prend deux coups d'horloge à un échantillon pour parcourir le bloc *min-max*. Puisque d_1 échantillons d'un signal brut sont analysés pour déterminer son minimum et son maximum, le temps à consommer pour déterminer le minimum et le maximum T_{minmax} d'un signal est $2d_1T_{clk}$. Ensuite, le temps T_{ajust} représente le temps nécessaire pour rapporter les signaux autour de 0. Il est égal à 2 coups d'horloge. Troisièmement, T_{div_norm} est le temps consommé par le sous-bloc *diviseur*. Comme le temps de latence du modèle *XSG* arithmétique assurant la division est de $16T_{clk}$, et qu'il existe un registre à l'entrée et la sortie de celui, le temps T_{div_norm} est égale à $(16+2)T_{clk}$. Enfin le temps global T_{filtre_norm} est exprimé par :

$$T_{filtre\_norm} = T_{filtre} + T_{norme} = \left(\frac{n}{2} - 1\right) T_{clk} + (2d_1 + 20)\, T_{clk}$$
$$= \left(2d_1 + \frac{n}{2} + 19\right) T_{clk} \tag{C.2}$$

C.2 Temps d'exécution pour le calcul de la matrice de covariance

Le temps d'exécution du modèle d'implémentation de la matrice de covariance T_{cov} est la somme des temps pour les opérations de multiplication matricielle (T_{mul_cov}) et division (T_{div_cov}). Tout d'abord, le sous-système réalisant la multiplication matricielle a une architecture parallèle, c'est-à-dire qu'il calcule tous les éléments de la matrice de covariance en même temps. Plus précisément, il calcule chaque élément de la matrice triangulaire supérieure et diagonale de la matrice de covariance. Le temps consommé pour le calcul de cette matrice est égal au temps de latence du bloc de multiplication *XSG* pour un échantillon (T_{Limul_cov}), multiplié par le nombre d'échantillons N_{sample} additionné au temps de latence initial du bloc *accumulator XSG* accumulant les échantillons T_{LiAcc_cov}.

$$T_{mul\_cov} = (T_{Limul\_cov} N_{sample} + T_{LiAcc\_cov})\, T_{clk} = (1 N_{sample} + 1)\, T_{clk}. \tag{C.3}$$

Le sous-système effectuant la division a une architecture série effectuant 4 divisions d'éléments de la matrice de covariance à la fois. Une fois que le bloc *diviseur* de *XSG* reçoit une donnée valide à son entrée à kT_{clk}, son entrée ne sera disponible pour une autre division qu'après 2 coups d'horloge. Ainsi la prochaine donnée peut être validée à son entrée à $(k+2)T_{clk}$. Aussi, les données divisées sont transmises à sa sortie après un temps de latence T_{Ldiv_cov} égale à $6T_{clk}$. À la fin de l'opération de division, les données passent par un registre T_{reg} avant d'être sauvegardées T_w .

$$T_{div\_cov} = \left((2\frac{N^2}{4} - 1) + T_{Ldiv\_cov} + T_{reg} + T_w\right) T_{clk} = \left(\frac{N^2}{2} + 7\right) T_{clk}. \tag{C.4}$$

En utilisant (C.3) et (C.4), le temps total consomé par le modèle d'implémentation de la matrice de covariance s'exprime comme suit

$$T_{cov} = \left(\frac{N^2}{2} + 8 + N_{sample}\right) T_{clk}. \tag{C.5}$$

C.3 Temps d'exécution de la transformée unitaire et son inverse

Selon le type de transformation à réaliser, le modèle implémentant la transformée unitaire et son inverse varie son temps d'exécution. En effet, le temps consommé pour effectuer une transformée unitaire T_{ut} est égale au double de celui de la transformée inverse T_{ut_inv}. En effet, cette dernière est obtenue en

effectuant une seule multiplication matricielle entre $\mathbf{V}_s$ et $\mathbf{U}$ contrairement à la transformée unitaire qui en requiert 2.

À chaque coup d'horloge, des éléments de la matrice à transformer sont lus pour calculer 2 éléments de la nouvelle matrice. Ainsi, pour parcourir une matrice $N \times N$, il faut $\frac{N^2}{2} - 1$ coups d'horloge. L'obtention de ces 2 éléments de la nouvelle matrice soumise à la transformation implique 3 opérations : lecture depuis les DPs A et B, opération arithmétique, et écriture dans les DPs A et B. Ces opérations consomment en temps chacunes un coup d'horloge T_{clk}. Par conséquent, le temps de calcul pour une transformée inverse d'une matrice $N \times N$ par le FPGA est :

$$T_{ut\_inv} = \left(\frac{N^2}{2} - 1\right) T_{clk} + (T_{clk} + T_{clk} + T_{clk}) = \left(2 + \frac{N^2}{2}\right) T_{clk}. \tag{C.6}$$

Enfin, l'expression de la transformation unitaire est déduite à partir de (C.6) comme ci-après :

$$T_{ut} = 2T_{ut\_inv} = \left(4 + N^2\right) T_{clk}. \tag{C.7}$$

C.4 Temps d'exécution de la décomposition EVD

La détermination du temps de calcul du modèle implémentant la décomposition EVD (T_{evd}) de la matrice de covariance est basée sur celle des sous-systèmes angles T_{ang}, rotation T_{rot} et ordonnancement T_{ord}. En effet, T_{evd} est égal à la somme des temps T_{ang}, T_{rot} et T_{ord}, multipliée par le nombre d'itérations $N \log N + 5$. Le détail du calcul de ces derniers est présenté ci-après.

Le bloc *angles* calcule $\frac{N}{2}$ angles utilisés dans l'opération de la rotation à chaque itération h. Sa structure interne illustrée à la figure 5.14 montre que pour calculer un angle, il faut additionner le temps de lecture depuis les DPs A et B des éléments de PD, le temps de calcul des opérandes à l'entrée du *Cordic* et le temps de calcul du *Cordic*. La figure C.1 montre la séquence temporelle pour le calcul des $\frac{N}{2}$ angles et le tableau C.1 dresse une description de ces différents temps.

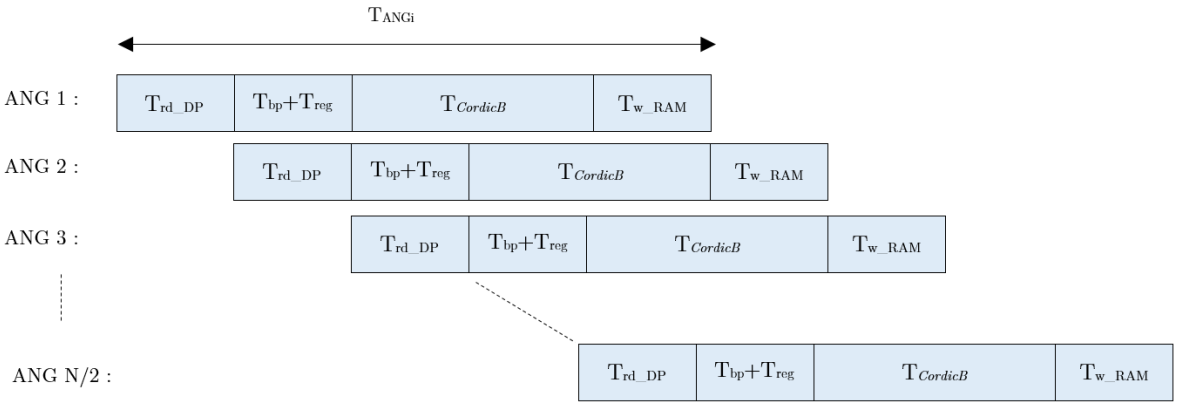


FIGURE C.1 – Répartition du temps d'exécution des tâches du bloc *angles* implémenté

Temps	Description
T_{rd_DP}	Temps nécessaire pour lire depuis les DPs A et B les données pour calculer l'angle. Avec les DPs, un seul coup d'horloge suffit pour lire les 3 éléments qu'il faut.
T_{bp}	Temps que consomme le sous-système *blocparamètre* inclus à la figure C.1 pour calculer le dénominateur et le numérateur de l'expression (4.12). Ce temps est inférieur à un cycle d'horloge.
T_{reg}	Temps mis par un signal pour passer de l'entrée d'un registre à sa sortie. Ce temps équivaut à un cycle d'horloge T_{clk}.
$T_{CordicB}$	Temps d'exécution du *Cordic* pour calculer l'arc-tangente. Il équivaut au nombre de bits des données à l'entrées du *CordicB* + 2.
T_{w_RAM}	Temps nécessaire pour sauvegarder un angle dans la RAM. Un seul coup d'horloge suffit pour cette opération.

TABLE C.1 – Description des différents temps pour le calcul des angles de rotation

En se basant sur le tableau C.1 et la figure C.1, l'expression du temps d'exécution du bloc *angles* implémenté pour une itération s'écrit :

$$T_{ang} = (\frac{N}{2} - 1)T_{clk} + T_{rd\_DP} + T_{bp} + T_{reg} + T_{CordicB} + T_{w\_RAM} = \left(\frac{N}{2} + n + 4\right) T_{clk}. \tag{C.8}$$

Tel que discuté à la section 5.5.2, pour déterminer les EVD, les PDs et les PNDs sont soumis à une double rotation et les PVs à une simple rotation réalisée par les *Cordics A1* et *A2* illustrés à la figure 5.17. À chaque itération, les processeurs PDs, PNDs et PVs se partagent les 2 *Cordics A1* et *A2*, dans une séquence bien définie montrée à la figure C.2.

Comme il est illustré à la figure C.2, les PDs et PNDs sont lus depuis les DPs A et B et transmis aux *Cordics A1* et *A2* dans un temps T_{rot1}. Ensuite, après un coup d'horloge T_{reg}, les PVs lus à leur tour depuis les DPs A et B sont transmis aux *Cordics A1* et *A2* dans un temps T_{PV}. Pendant ce temps, les premiers résultats aux sorties des *Cordics A1* et *A2* correspondants à la rotation des PDs et PNDs sont sauvegardés dans des FIFOs. Cela en attendant que les entrées des *Cordics* occupées à recevoir les derniers PVs soient de nouveau disponibles.

À la fin de la transmission des PVs, les entrées des *Cordics A1* et *A2* sont laissées inutilisées de nouveau pendant un coup d'horloge T_{reg}. Ensuite, les résultats de la première rotation sauvegardés dans les FIFOs sont transmis aux entrées des *Cordics A1* et *A2* pour une seconde rotation. Durant ce processus, les résultats de la rotation des PVs sont sauvegardés dans les FIFOs en attente du premier résultat de la seconde rotation des PDs et PNDs. Le temps écoulé entre l'entrée des *4* éléments du premier processeur $\text{PD}_{1,1}$ de la seconde rotation et leur sortie dans les *Cordics A1* et *A2* est $T_{CordicA}$. Ce dernier est égal au nombre de bits ($n$bits) des données transmis aux *Cordics* additionné de 2. L'opération d'ordonnancement commence dès la validation à la sortie des *Cordics* des résultats correspondants à la rotation des 4 éléments de $\text{PD}_{1,1}$. Ainsi, à partir de ce qui précède, le temps consommé par le FPGA

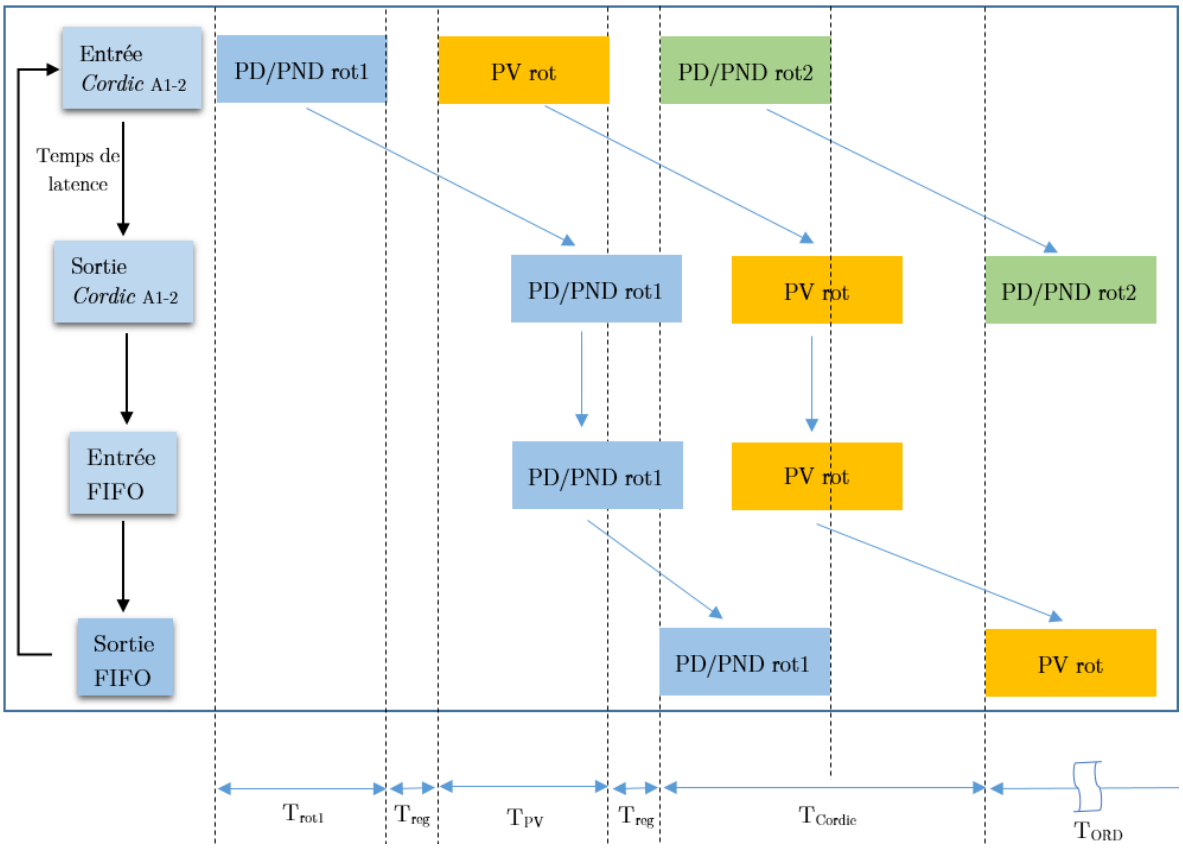


FIGURE C.2 – Séquence temporelle des rotations des processeurs PD, PND et PV pour le calcul des EVD

pour effectuer les rotations des processeurs au cours d'itération est donné par l'équation suivante :

$$T_{rot} = T_{rot1} + T_{reg} + T_{PV} + T_{CordicA} = \left(\frac{N^2}{4} + 1 + \frac{N^2}{4} + 1 + n + 2\right) T_{clk}$$
$$T_{rot} = \left(\frac{N^2}{2} + n + 4\right) T_{clk}. \tag{C.9}$$

En ce qui concerne l'opération d'ordonnancement des matrices $\mathbf{R}_{xx}^{(h+1)}$ et $\mathbf{V}^{(h+1)}$, elle se résume à la sélection des multiplexeurs et à la génération de nouvelles adresses d'emplacements des éléments dans les DPs A et B. Ces 2 tâches nécessitent moins d'une période d'horloge T_{clk} pour les 4 éléments d'un processeur. Par conséquent, le temps T_{ord} est assimilable au temps d'écriture des éléments d'un processeur T_{w_DP} dans les DPs A et B et s'écit comme suit :

$$T_{ord} = T_{w\_DP} = \frac{N^2}{4} T_{clk}. \tag{C.10}$$

Enfin, le temps d'exécution pour la décomposition EVD de la matrice de covariance par le FPGA T_{evd}

est donné par l'équation ci-après

$$T_{evd} = (T_{ang} + T_{rot} + T_{ord})\, H$$
$$T_{evd} = \left(\frac{3}{4}N^2 + \frac{1}{2}N + 2n + 8\right) H T_{clk} \tag{C.11}$$

où H est égale au nombre d'itérations $N \log N + 5$.

C.5 Temps d'exécution du bloc *tri*

Le temps consommé par le FPGA pour trier (T_{tri}) en ordre décroissant les valeurs propres et leurs vecteurs propres associés est variable. En effet, il dépend de l'ordre dans lequel les valeurs propres seront à la fin de la décomposition EVD. Puisque cet ordre est aléatoire, le pire scénario sera considéré pour déterminer le temps d'exécution du bloc *tri* implémenté. Ainsi, l'ordre des valeurs et vecteurs propres considéré avant l'exécution du tri est croissant.

Cependant, les tâches durant le premier balayage du tri à bulle suivent la séquence temporelle suivante :

1. Deux valeurs successives λ_i et λ_{i+1} sont lues simultanément parmi les N valeurs propres non ordonnées. Une est considérée comme valeur de référence et l'autre comme valeur à comparer. Un seul coup d'horloge suffit pour cette lecture, puisqu'elle est faite depuis les DPs A et B (T_{rd_DP}).
2. La valeur propre comparée devient la nouvelle valeur de référence, car l'hypothèse fixée spécifie $\lambda_i < \lambda_{i+1}$ et une nouvelle valeur propre à comparer est lue. Le temps écoulé correspond à un temps registre (T_{reg}) et temps de lecture (T_{rd_DP}).
3. La séquence 2 est répétée jusqu'à la comparaison entre λ_{N-1} et λ_N. Ainsi, la durée d'exécution est égale à $(T_{rd\_DP} + T_{reg})\,(N-1)$.
4. La permutation des emplacements mémoires des colonnes contenant la valeur propre maximale et la valeur propre minimale est réalisée. Cette tâche requiert un temps de lecture T_{rd_DP} et un temps d'écriture T_{w_DP} pour chaque élément des colonnes. Soit une durée égale à $N(T_{rd\_DP} + T_{w\_DP})$.
5. Pour le changement de la ligne des valeurs propres dans les colonnes permutées, le temps consommé est de $2(T_{rd\_DP} + T_{w\_DP})$.
6. Désormais, le nombre de valeurs propres à ordonner est décrémenté $(N = N-1)$.
7. Les séquences 1 à 6 sont répétées jusqu'à ce que les valeurs et vecteurs propres soient classés en ordre décroissant.

De ce qui précède, l'équation de la durée d'exécution du tri par le FPGA est déduite et exprimée ci-dessous :

$$
\begin{aligned}
T_{tri} &= \left(T_{rd\_DP} + N(T_{rd\_DP} + T_{w\_DP}) + 2(T_{rd\_DP} + T_{w\_DP})\right)(N-1) + \sum_{k=1}^{N-1}(T_{rd\_DP} + T_{reg})k \\
&= T_{clk}\left[(2N+5)(N-1) + \sum_{k=1}^{N-1} 2k\right] = T_{clk}\left[(2N+5)(N-1) + (N-1)N\right] \\
&= \left[(N-1)(3N+5)\right]T_{clk}. \qquad \text{(C.12)}
\end{aligned}
$$

C.6 Temps d'exécution du bloc de détection du nombre de sources

La figure C.3 montre la séquence temporelle des tâches réalisées par le bloc de détermination du nombre de sources. Ces différentes tâches sont détaillées à la section 5.6.2. Cette séquence temporelle est décrite comme suite :

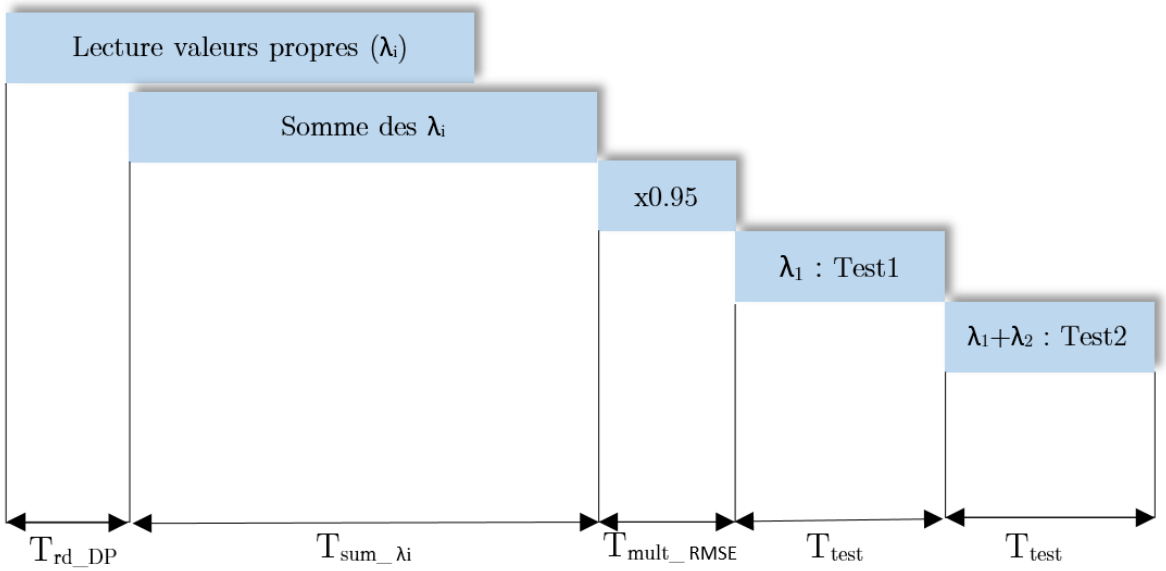


FIGURE C.3 – Séquence temporelle des tâches du bloc de détection du nombre de sources

1. La lecture des N valeurs propres sauvegardées dans les DPs A et B et chaque valeur propre nécessite un temps $T_{rd\_DP} = T_{clk}$;
2. L'accumulation des valeurs propres débute dès que la première valeur propre est disponible à l'une des sorties des DPs A et B. Le temps nécessaire pour accumuler les N valeurs propres $T_{sum\_\lambda i}$ est de NT_{clk} ;
3. La somme totale des valeurs propres est multipliée par 0.95 en une durée $T_{mult\_RMSE} = T_{clk}$;
4. La durée d'un test T_{test} comprend un temps de lecture d'une valeur propre et un autre pour la comparaison et l'incrémentation du compteur de nombre de sources. Ce temps est variable, car il dépend du nombre de sources.

Le temps total consommé par le FPGA pour déterminer le nombre de sources est donné par l'équation ci-après :

$$\begin{aligned} T_{nsrc} &= T_{rd\_DP} + T_{sum\_\lambda i} + T_{mult\_RMSE} + MT_{test} \\ T_{nsrc} &= (2+N+M)\,T_{clk} \end{aligned} \tag{C.13}$$

où M est le nombre estimé de sources.

C.7 Temps d'exécution du bloc de calcul du vecteur de balayage

La séquence temporelle des tâches réalisées par le bloc de calcul des vecteurs $\mathbf{a}(\theta)$ présenté à la section 5.7.1 est illustrée à la figure C.4. Tout d'abord, le temps de calcul de $\phi(\theta)$ composé de T_{start}, période de latence entre le signal de déclenchement du bloc et la validation de la phase à l'entrée du *Cordic*. À ce temps, il faut ajouter le temps de traitement du *Cordic* T_{Cordic} en mode *cos-sin*. Le $\phi(\theta)$ calculé est sauvegardé dans un registre (T_{reg}). Ensuite, il est multiplié par une série de nombre de 0 à $N-1$ et autres variables comme il est exprimé à l'équation (2.7). Ceci forme les opérandes à l'entrée du *Cordic* ($T_{opt} = T_{clk}$). Une fois que chaque opérande est calculé, il est immédiatement transmis à l'entrée du *Cordic* car, l'entrée de celui-ci est disponible à chaque coup d'horloge. Ainsi, il faut une durée NT_{clk} après le calcul de ϕ pour avoir la dernière opérande à l'entrée du *Cordic*. Enfin, après $T_{Cordic} + T_{reg}$, le dernier élément du vecteur **a** est obtenu.

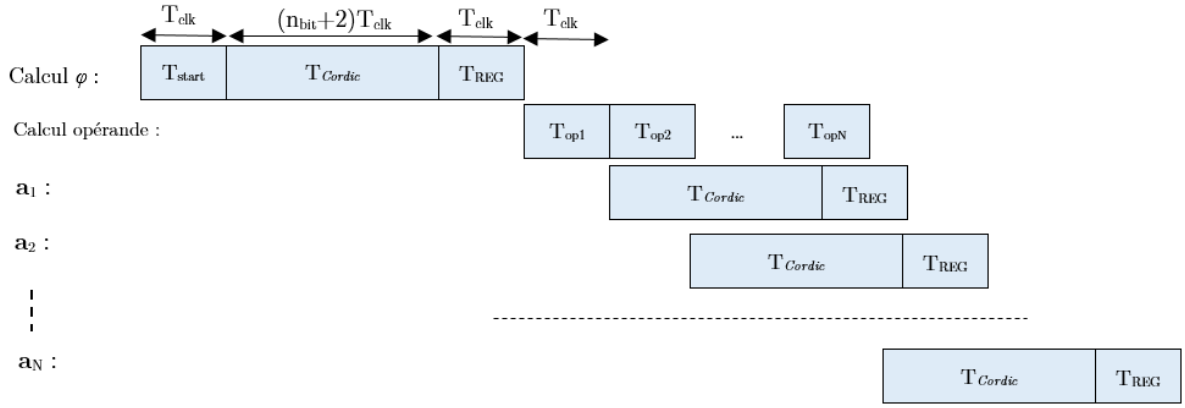


FIGURE C.4 – Séquence temporelle des tâches du bloc de calcul du vecteur **a**

On peut alors écrire l'équation du temps d'exécution du calcul du vecteur **a**, comme suit :

$$\begin{aligned} T_{vecta} &= T_{start} + T_{Cordic} + T_{reg} + NT_{opt} + T_{Cordic} + T_{reg} = (2+(n+2)+1+N+(n+2)+1)T_{clk} \\ T_{vecta} &= (8+2n+N)\,T_{clk} \end{aligned} \tag{C.14}$$

où n, faut-il le rappeler, est le nombre de bits sur lesquels s'effectuent les calculs.

C.8 Annexe C : Temps d'exécution du bloc de calcul de la diagonale de la matrice G et du bloc de calibration

Le bloc de calcul de la diagonale de la matrice **G** est présenté à la section 5.7.2. Le temps d'exécution de ce bloc par le FPGA dépend de la séquence temporelle des tâches montrées à la figure C.5. Cette séquence débute par le temps de calcul du vecteur **a**, auquel est retranché le temps de sortie de ses N éléments. Ceci, car les éléments de **a** ne sont pas enregistrés dans une mémoire à la fin de leur calcul, mais plutôt utilisés directement dans des calculs arithmétiques. En effet, la lecture des éléments du vecteur propre principal $\mathbf{v}_1$ débute lorsque le premier élément de **a** est disponible. Un registre permet de synchroniser les éléments du vecteur **a** et ceux de $\mathbf{v}_1$ ($T_{reg} = T_{clk}$). Par la suite, à chaque coup d'horloge, un élément de chaque vecteur **a** et $\mathbf{v}_1$ est transmis au bloc de multiplication. Après une période d'horloge, le résultat de la première multiplication est aussitôt transmis à l'entrée du bloc *XSG* effectuant la division. Comme ce bloc est disponible $2T_{clk}$ plus tard à chaque fois qu'une donnée est validée à son entrée, les autres résultats de la multiplication passeront par une FIFO. Il faut $2(N-1)T_{clk}$ pour transmettre tous les résultats de la multiplication au bloc *XSG* de division. Enfin, la dernière division est réalisée en $6T_{clk}$ auquel s'ajoute le temps d'écriture du dernier élément de la diagonale de la matrice **G** dans une RAM (T_{clk}). Ainsi, l'équation du temps d'exécution du bloc de calcul de la diagonale de la matrice **G** s'écrit :

$$\begin{aligned} T_{matG} &= (T_{vecta} - NT_{clk}) + T_{clk} + T_{clk} + 2(N-1)T_{clk} + 6T_{clk} + T_{clk} = T_{vecta} + (7+N)T_{clk} \\ T_{matG} &= (15 + 2n + 2N)\, T_{clk}. \end{aligned} \tag{C.15}$$

En ce qui concerne le bloc de calibration, l'agencement et la durée de ses tâches sont montrés à la figure C.6. Ces tâches sont au nombre de trois et sont décrites dans le tableau C.2.

Temps	Description
T_{lect_cal}	Temps de lecture de 4 éléments de la diagonale de la matrice **G** depuis une mémoire RAM ($4T_{clk}$) et lecture de 4 éléments de la matrice $\mathbf{R}_{xx}$ depuis DP A et B (T_{clk}).
$T_{mult\_complex}$	Le temps de latence des sous-systèmes *Mult_mat* effectuant la multiplication complexe égale à $4T_{clk}$. Comme il est montré à la figure 5.23, deux sous-systèmes reliés en série permettent de calibrer un élément de la matrice de covariance. Ainsi, il faut considérer 2 fois $T_{mult\_complex}$ dans le temps d'exécution.
T_{w_DP}	Temps d'écriture des 4 éléments calibrés dans les DPs A et B. Un seul coup d'horloge est nécessaire.

TABLE C.2 – Description des différents temps pour la calibration de la matrice de covariance

En fait, la séquence temporelle illustrée à la figure C.6 est la durée d'un cycle de traitement du FPGA pour calibrer 4 éléments de la matrice de covariance $\mathbf{R}_{xx}$ de taille $N \times N$. Cependant, il faut $\frac{N^2}{4}$ fois cette

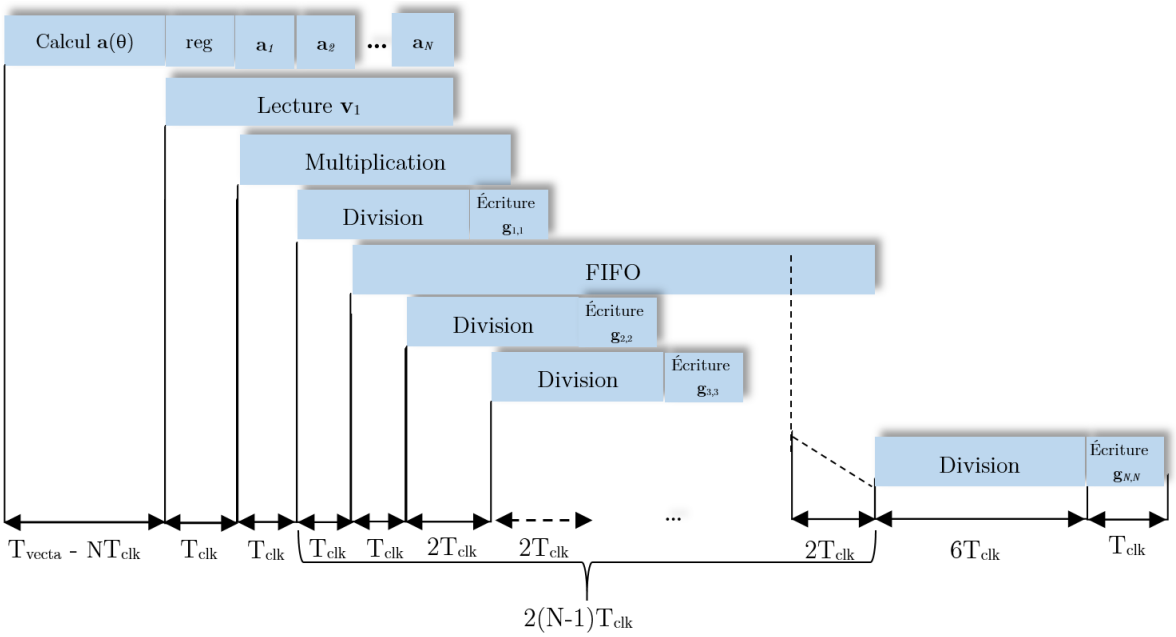


FIGURE C.5 – Séquence temporelle des tâches du bloc de calcul de la diagonale de la matrice **G**

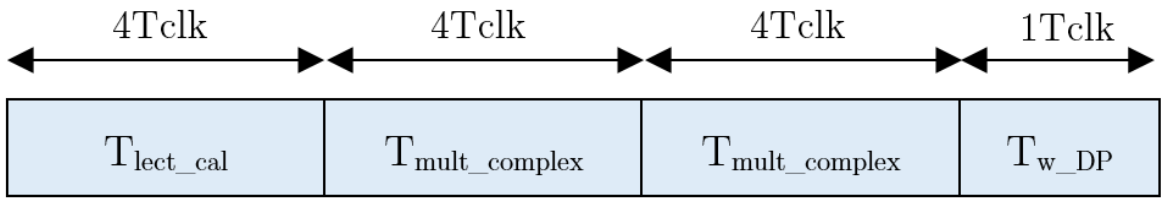


FIGURE C.6 – Séquence temporelle des tâches du bloc de calibration de 4 éléments de la matrice $\mathbf{R}_{xx}$

durée pour compléter la calibration des $N \times N$ éléments de la matrice $\mathbf{R}_{xx}$. Ainsi, l'équation exprimant cette durée est la suivante :

$$T_{cal} = \frac{N^2}{4}\left(T_{lect\_cal} + 2T_{mult\_complex} + T_{w\_DP}\right) = \frac{N^2}{4}\left(4+8+1\right)T_{clk}$$
$$T_{cal} = \left(\frac{13}{4}N^2\right)T_{clk}. \tag{C.16}$$

C.9 Temps d'exécution du bloc de calcul du pseudo-spectre de MUSIC

La figure C.7 donne un aperçu des différentes tâches réalisées par le bloc pseudo-spectre implémenté dans le FPGA, ainsi que leur déroulement dans le temps. La séquence débute par le temps de calcul de la matrice de projection $\mathbf{P}_n$ (T_{Pn}). Pour le calcul d'un élément de la matrice de projection, le processus dans le temps est le suivant :

1. Le temps de lecture d'éléments de même ligne pour les M vecteurs propres sources. Puisqu'il

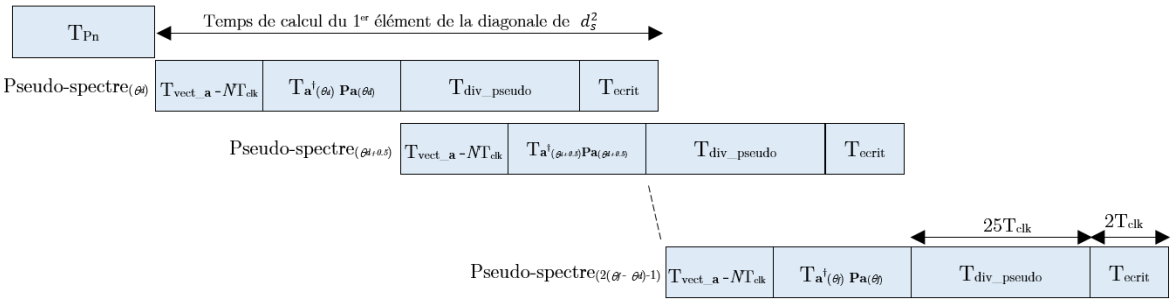


FIGURE C.7 – Séquence temporelle des tâches du pseudo-spectre de MUSIC

faut un coup d'horloge par élément, la durée de lecture est égale à MT_{clk} ;

2. À chaque fois qu'un élément des vecteurs propres sources est lu, il est multiplié par lui-même. Sachant que le temps de latence pour une multiplication est égal à un coup d'horloge, le temps de la multiplication est alors égal à MT_{clk} pour les M éléments lus ;
3. De même, chaque fois qu'une multiplication est terminée, le résultat est accumulé. Sachant que le temps de latence de l'accumulateur est égal à T_{clk}, le temps total pour l'accumulation est donc égal MT_{clk} ;
4. À la fin de l'accumulation, un coup d'horloge suffit pour enregistrer le nouvel élément de la matrice $\mathbf{P}_n$.

En outre, les 3 premières tâches ayant chacune une durée de MT_{clk} sont réalisées en parallèle, mais sont déclenchées l'une à la suite de l'autre et espacées un coup d'horloge chacune. Quant à la 4^e tâche, elle est réalisée $(2+M)T_{clk}$ plus tard, après que toutes les précédentes soient complétées. De plus, elle ne dure qu'un coup d'horloge. Ainsi, le temps nécessaire pour calculer un élément de la matrice $\mathbf{P}_n$ est égale $(3+M)T_{clk}$. Enfin, le temps total consommé par le FPGA pour calculer la matrice $\mathbf{P}_n$ de taille $N \times N$ est donné par l'équation suivante :

$$T_{Pn} = N^2 \left(3+M\right) T_{clk}. \tag{C.17}$$

La suite de la séquence temporelle du pseudo-spectre est une succession du calcul des éléments de la diagonale de d_s^2. Le temps de calcul d'un élément de ce dernier est donné par l'addition des temps des tâches suivantes :

1. Le temps de calcul du vecteur $\mathbf{a}$ T_{vecta} auquel il faut retrancher NT_{clk}, le temps de sortie des éléments de $\mathbf{a}$;
2. Le temps de la double multiplication ($T_{doublmult\_a}$) de $\mathbf{P}_n$ par $\mathbf{a}^\dagger$ et $\mathbf{a}$;
3. Le temps de latence du bloc *XSG* effectuant la division T_{div_pseudo} qui équivaut à $25T_{clk}$;

4. Le temps d'écriture dans la mémoire *SDRAM DDR3* externe au FPGA équivalant à un coup d'horloge.

La séquence temporelle de l'opération de la double multiplication de $\mathbf{P}_n$ par le vecteur $\mathbf{a}$ et sa transposée est montrée à la figure C.8 . Tout d'abord, le temps d'une multiplication du vecteur $\mathbf{a}$ transposé à la colonne i de $\mathbf{P}_n$ est égal $(N+1)T_{clk}$. Le résultat de cette multiplication est multiplié par l'élément i de $\mathbf{a}$ et accumulé pour une durée égale à $2T_{clk}$. Ainsi, pour prendre en compte toutes les colonnes, il faut multiplier la somme de ces deux durées par le nombre de colonnes de $\mathbf{P}_n$. À ce temps, il faut ajouter un coup d'horloge pour le déclenchement du calcul d'un nouveau vecteur $\mathbf{a}$ et la validation de l'opération de division. On peut alors exprimer le temps de calcul de la double multiplication de la matrice de projection $\mathbf{P}_n$ par les vecteurs $\mathbf{a}^\dagger$ et $\mathbf{a}$ par l'équation ci-après :

$$T_{doublmult\_a} = \left(N(N+3)+1\right) T_{clk}. \tag{C.18}$$

Comme on peut le voir à la figure C.7, le calcul du prochain élément de la diagonale de d_s^2 débute lorsque l'opération de double multiplication $\mathbf{a}^\dagger \mathbf{P}_n \mathbf{a}$ du calcul du précédent élément de la diagonale de d_s^2 est achevée. Ainsi, le temps qu'il faut au dernier élément de d_s^2 pour être disponible à l'entrée du bloc *XSG* de division est égal à la multiplication de la somme de T_{vecta} et $T_{doublmult\_a}$ par le nombre d'éléments de la diagonale de d_s^2, le tout additionné à T_{Pn}. La taille de la diagonale de d_s^2 est reliée à la longueur de la plage d'angle $[\theta_d, \theta_f]$ à balayer. Enfin, en ajoutant la durée de la division (T_{div_pseudo}) et le temps d'écriture du résultat de la division au temps d'entrée du dernier élément au bloc *XSG* de division, on obtient le temps total d'exécution du bloc du pseudo-spectre par le FPGA. Ce dernier est exprimé par l'équation suivante :

$$\begin{aligned} T_{pseudo\_MUSIC} &= T_{Pn} + Q\left[(T_{vecta} - NT_{clk}) + T_{doublmult\_a}\right] + T_{div\_pseudo} + T_{ecrit} \\ &= N^2(3+M)T_{clk} + Q\left[(8+2n)T_{clk} + (N(N+3)+1)T_{clk}\right] + 25T_{clk} + 2T_{clk} \\ &= \left[Q\left(N(N+3)+2n+9\right) + N^2(3+M) + 27\right] T_{clk} \end{aligned} \tag{C.19}$$

où Q représente le nombre de pas pour balayer la plage d'angles au complet.

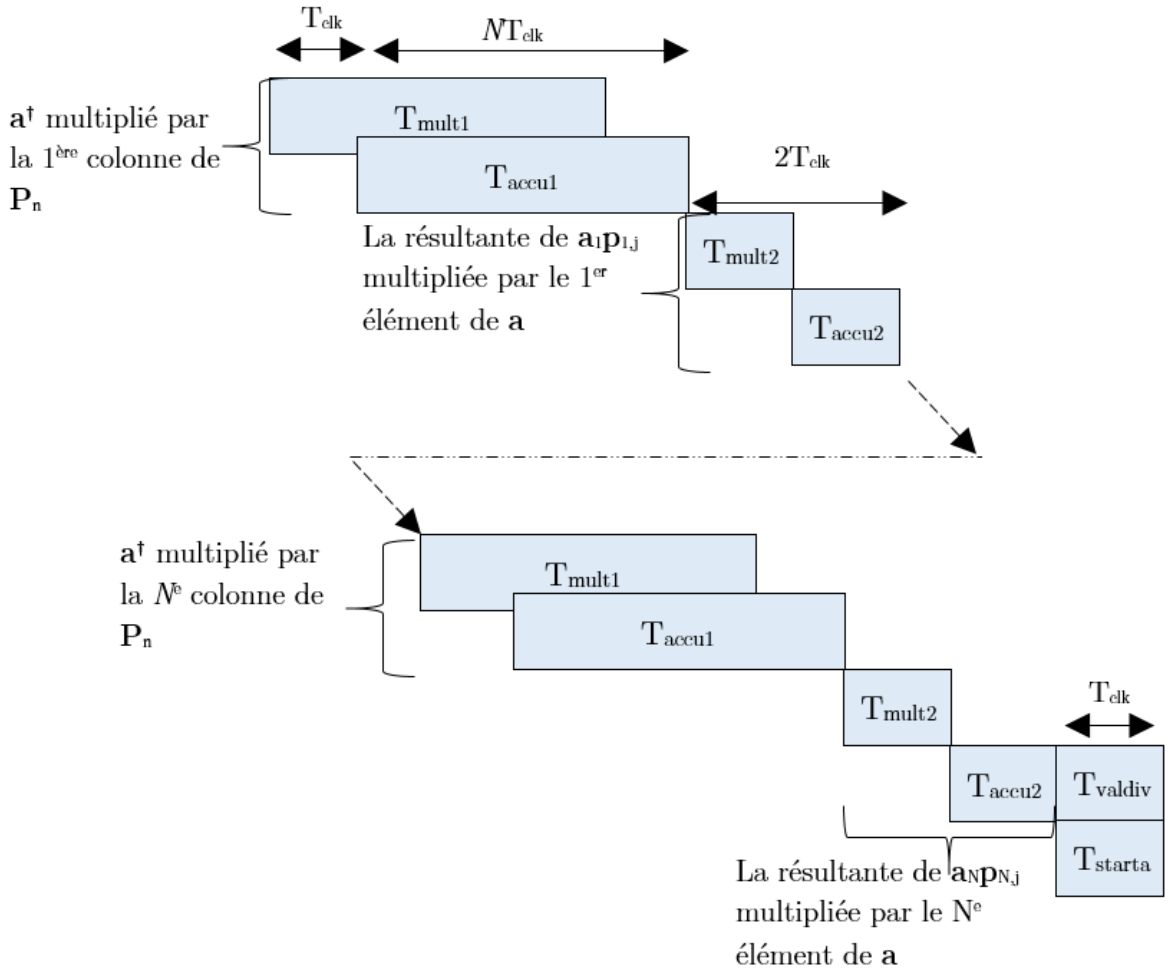


FIGURE C.8 – Séquence temporelle des tâches de la double multiplication de la matrice de projection $\mathbf{P}_n$ par les vecteurs $\mathbf{a}^\dagger$ et $\mathbf{a}$

Bibliographie

[1] Grenier Dominic. Antennes et propagation radio. http://w3.gel.ulaval.ca/~dgrenier/ap_notes-e.pdf, 2016. Manuel de cours.

[2] Arbenz Dr. Peter. Numerical methods for solving large scale eigenvalue problems. http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter3.pdf.

[3] Hung E. Matrix-construction calibration method for antenna arrays. *IEEE Transactions*, (3) :819–828, July 2000. Aeorospace and Electronics Systems.

[4] Park H.G, Jung J.H., Oh H.S., and Kyeong M.G. Model based antenna array calibration for smart antenna arrays. *Electronics Letters*, 38(15) :771–772, July 2002.

[5] Bertrand Hugo, Grenier Dominic, and Roy Sebastien. Experimental antenna array calibration with adaptive linear neuron (adaline) network. *IEEE*, 2006.

[6] Huarng Keh-Chiarng and Yeh Chien-Chung. A unitary transformation method for angle-of-arrival estimation. *IEEE Transactions*, 1991.

[7] Mathworks. Introduction to the filter design and analysis tool. http://www.mathworks.com/help/signal/examples/introduction-to-the-filter-design-and-analysis-tool-fdatool.html. Accessed : 2016-01-01.

[8] Kim Minseok. *Hardware implementation of Signal Processing in Smart Antenna Systems for High Speed Wireless Communication*. Thesis, Yokohama National University, Department of Electrical and Computer Engineering, Dec 2004.

[9] H.G Park and S.C. Bang. Model based antenna array calibration for digital beamforming systems. *IEEE proceedings*, 2 :867–870, Apr 2003. Vehicular Technology Conference.

[10] J. Pierre and M. Kaveh. Experimental performance of calibration and direction-finding algorithm. *IEEE*, 2 :1365–1368, Apr 1991. Acoustics, Speech, and Signal Processing.

[11] S. Unnikrishna Pillai and Byung Ho Kwon. Fonvard/backward spatial smoothing techniques for coherent signal identification. *IEEE Transactions*, 37 :8–15, January 1989. Acoustics speech and signal processing.

[12] P. Brent Richard and T. Luk Franklin. The solution of singular-value and symmetric eigenvalue problems on miultiprocessor array's. *SIAM Journal on Scientific and Statistical Computing*, 1985.

[13] P. Brent Richard, T. Luk Franklin, and C. Van Loan. Computation of singular value decomposition using mesh-connected processors. *VLSI and Computer Systems*, 1985.

[14] R.O. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions*, 34 :276–280, Mar 1986. Antennas Propag.

[15] M. Kay Steven. *Modern Spectral Estimation : Theory and Application*. Advanced monographs. PTR Prentice Hall, 1988.

[16] Xilinx. Xilinx system generator user guide. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_user.pdf, Dec 2009.

[17] Xilinx. Logicore ip cordic v5.0. http://www.xilinx.com/support/documentation/ip_documentation/cordic/v5_0/ds858_cordic.pdf, Oct 2011.

[18] Xilinx. Logicore ip fir compiler v6.3. http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v6_3/ds795_fir_compiler.pdf, Oct 2011.

[19] Chen Y.-M., Lee J.-H., Yeh C.-C., and J. Mar. Bearing estimation without calibration for randomly pertubeted array. *IEEE Transactions*, 39(1) :194–197, Jan 1991. signal Processing.