RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département d'Electronique
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaïque

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'ingénieur d'état en Electronique

Thème :

IMPLEMENATION SUR FPGA D'UN DECODEUR SPATIO-TEMPOREL POUR LES SYSTEMES MIMO DE COMMUNICATION SANS FILS.

DJELILI Amina

Sous la direction de

Mr. M. TAGHI

Présenté et soutenue publiquement le 18 Juin devant le jury composé de :

Président	MCA. R. SAADOUN	(ENP)
Examinateur	Pr. D. BERKANI	(ENP)
Promoteur	MAA. M. TAGHI	(ENP)

ENP 2017

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département d'Electronique
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaïque

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'ingénieur d'état en Electronique

Thème :

IMPLEMENATION SUR FPGA D'UN DECODEUR SPATIO-TEMPOREL POUR LES SYSTEMES MIMO DE COMMUNICATION SANS FILS.

DJELILI Amina

Sous la direction de

Mr. M. TAGHI

Présenté et soutenue publiquement le 18 Juin devant le jury composé de :

Président	MCA. R. SAADOUN	(ENP)
Examinateur	Pr. D. BERKANI	(ENP)
Promoteur	MAA. M. TAGHI	(ENP)

ENP 2017

A ma mère et mon père

A mes frères et ma soeur

A mes amis

Remerciements

Ce mémoire de fin d'études à été effectué au sein du département d'Electronique de l'Ecole Nationale Polytechnique d'Alger.
Je tiens à remercier mon promoteur M.Taghi qui m'a soutenu tout au long de la réalisation de ce projet.
Je remercie tous les professeurs et enseignants dont j'ai eu l'honneur de côtoyer et qui m'ont fourni les bases nécessaires pour accomplir ce travail durant mon cursus.
Mes plus vifs remerciements s'adressent aussi à tout le cadre professionnel et administratif de L'école Polytechnique.
Mes remerciements vont enfin à toute personne qui a contribué de près ou de loin à l'élaboration de ce travail.

ملخص

يتمحور هذا العمل حول محاكاة و تثبيت خوارزمية فك التشفير الاموتي علي بطاقة ف ب ح اي. هذه دراسة لتقنيات التشفير و فك التشفير الزماني المكاني بالكتل لأنظمة الميمو و يليه محاكاة و تثبيت خورزمية فك التشفير ألاموتي و الدوائر المرتبطة به علي بطاقة ال ف بي جي اي.

الكلمات المفتاحية: الكتل الزمنبة المكانية، ألاموتي ، فك التشفير ا ألاموتي.

Abstract

This work revolves around the simulation and the implementation on FPGA of an Alamouti decoder for MIMO communications systems.
This is a study of the techniques of space-time block coding & decoding of the MIMO systems followed by the implementation of the decoding of Alamouti and its associated circuits on the FPGA ML501 Board.

Keys words: space-time block, Alamouti, Alamouti décoder.

Résumé

Ce travail s'articule autour de la simulation et l'implémentation sur FPGA d'un décodeur d'Alamouti pour les systèmes de communications MIMO.
Il s'agit d'une étude des techniques de codage / décodage spatio-temporels en bloc des sytèmes MIMO suivie de l'implémentation du décodeur d'Alamouti et des circuits associés sur la carte FPGA ML501.

Mots clé : Spatio-temporel en bloc, Alamouti, décodeur d'Alamouti.

Table des matières

Table des figures

Liste des tableaux

Acronymes

AWGN	Average White Gaussian Noise
BER	Bit rate error
BPSK	Binary Phase Shift Keying
CDM	Codeword Distance Matrix
CGD	Gain de codage (Cumulative Gain Distance)
CSI	Channel State Information
CST	Code Spatio Temporels
FER	Frame Error Rate
ISI	Inter Symbols Interference
LOS	Line Of Sight
M-ASK	M - Amplitude Shift Keying
MIMO	Multiple Input Multiple Output
MISO	Multiple Input Single Output.
ML	Maximum Likelihood
MRC	Maximum Ratio Combining
MV	Maximum de Vraisemblance
OFDM	Orthogonal Frequency Division Multiplexing
QPSK	Quaternary Phase Shift Keying
QAM	Quadrature Amplitude Modulation.
SIMO	Single Input Multiple Output.
SISO	Single Input Single Output
SNR	Signal to Noise Ratio
STBC	Space Time Bloc Code
STC	Space Time Coding

Notations et variables utilisées

a^*	Conjugué du scalaire a.
A	Matrice.
i	Nombre complexe défini par $i^2 = -1$.
I_n	Matrice identité de dimension nxn.
A^H	la matrice Hermitienne conjuguée A.
$\otimes$	Produit de Kronecker.
$\|a\|$	Module du scalaire a.
n	Nombre d'antennes de transmission.
p	Nombre de périodes de transmission.

Introduction générale

Les systèmes de communications modernes doivent satisfaire les exigences des usagers en matière de débit de transmission et de qualité de service.
Dans cette optique, les systèmes de transmission de type MIMO, comportant plusieurs antennes à l'émission et à la réception, sont considérés comme étant des techniques incontournables.L'efficacité spectrale potentielle de tels systèmes bien plus élevée que celle des systèmes mono-antenne et leur robustesse face aux évanouissements du canal grâce à une meilleure exploitation de la diversité améliorant ainsi la qualité de la transmission rend les systèmes MIMO très attractifs.

L'utilisation d'antennes multiples dans les systèmes de communications a commencé avec l'application de techniques de diversité spatiale de réception pour combattre les évanouissements dans ces Canaux. Ces systèmes étaient constitués d'une seule antenne en émission et de plusieurs antennes en réception. En recombinant les signaux résultant de ces nombreux canaux, il est alors possible d'accroître la robustesse du système. En 1998, Alamouti propose une technique très simple de diversité de transmission dont les performances sont équivalentes à celles de systèmes avec diversité spatiale de réception. Basés sur un codage en bloc des données, cette technique sera baptisée codage spatio-temporel en bloc (SpaceTime Block Coding, STBC). A la suite de ces travaux, des chercheurs ont proposé de nombreux schémas de STBC permettant de combiner diversité spatiale d'émission et de réception pour différentes configurations d'antennes.

Le travail exposé dans ce mémoire consiste à élélaborer les techniques de décodage spatio-temporels dans les systèmes de communication MIMO et à implémenter sur FPGA un décodeur basé sur la méthode d'Alamouti.

Plan du mémoire

Ce mémoire est organisé en 3 chapitres.

Le premier chapitre expose les notions de base nécessaires pour aborder les autres chapitres du manuscrit. Nous décrivons les caractéristiques et les notions associées à un canal radio mobile, puis nous définissons un modèle du système MIMO et on se limitera au cas de deux antennes pour aborder le théorème d'Alamouti. Les techniques de transmission MIMO (2 X 2) à l'émission et à la réception, associées à un codage de canal, sont abordées par la suite. Dans le cadre de ce travail, nous partons de l'hypothèse que le canal de

transmission est inconnu puis il est estimé à la réception. On décrira par la suite l'architecture que nous avons spécifié pour le récepteur MIMO.

Le deuxième chapitre aborde les aspects algorithmiques du codage et décodage d'Alamouti. On vérifiera le fonctionnement de notre architecture en simulant son fonctionnement sur MATLAB et Modelsim.

Dans le dernier chapitre on présentera brièvement la carte FPGA utilisée (FPGA ML 501) puis on exposera les resultats obtenu après l'implementation.

Chapitre 1

Etude théorique

1.1 introduction

Dans ce premier chapitre nous posons le cadre de notre travail et nous présentons les différentes notions utilisées dans les chapitres à venir. Nous présentons dans un premier temps un canal radio-mobile et les différentes caractéristiques qui lui sont associées. Nous définissons ensuite un modèle du système multi-antennes : MIMO en abordera ensuite le cas de deux antennes en emission et en réception. Une chaîne de transmission numérique permet le transport, en bande de base, d'information numérique entre un émetteur et un récepteur à travers un canal de transmission. Nous nous intéressons aux systèmes MIMO nécessitant uniquement la connaissance du canal à la réception. Sous cette hypothèse, le système MIMO emploie un codage espace-temps qui exploite les dimensions spatiale et temporelle apportées par le canal MIMO. Dans ce système, l'émetteur comprend les fonctions de codage de canal, codage espace temps.Quant au récepteur, il comprend les fonctions symétriques, à savoir : la détection MIMO, le décodage de canal.

1.2 Canal Radio Mobile

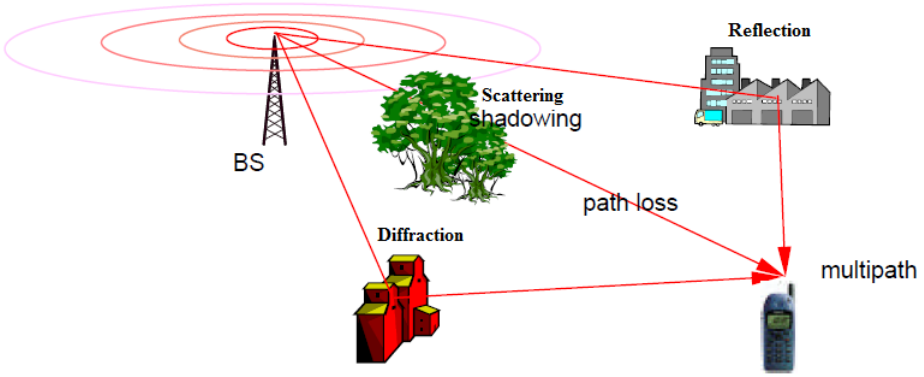

FIGURE 1.1 – Mécanismes de Propagation du signal de la station de base au mobile

Le signal transmis peut suivre plusieurs chemins avant d'atteindre le récepteur.Ces trajets multiples sont causés par la diffraction, réfraction, la dispersion,déplacement du mobile..etc

d'où le signal suivant ces trajets peut subir des évanouissements en amplitude ou en phase selon l'équation qui suit [5] :

$$Y = H.X + W \tag{1.1}$$

tel que :
H : évanouissement (représente le canal).
X : signal transmis.
W : le bruit AWGN dû aux rayonnements captés par les antennes, les interférences éventuelles entre utilisateurs et le bruit généré par les composants électroniques.

En présence d'un large nombre d'évanouissements, le canal radio-mobile peut être modélisé par un processus aléatoire Gaussien complexe (théorème central limite).

1.3 Gain de puissance :

C'est l'augmentation moyenne du rapport signal sur Bruit SNR à la réception. La connaissance du canal à l'émission ou à la réception résulte sur la combinaison cohérente des signaux à la reception.Si le récepteur connait le canal on dit gain d'émission si non on dit gain de réception.
Généralement, les systèmes à plusieurs antennes nécessitent une connaissance parfaite sur le canal soit au niveau du récepteur ou les deux.

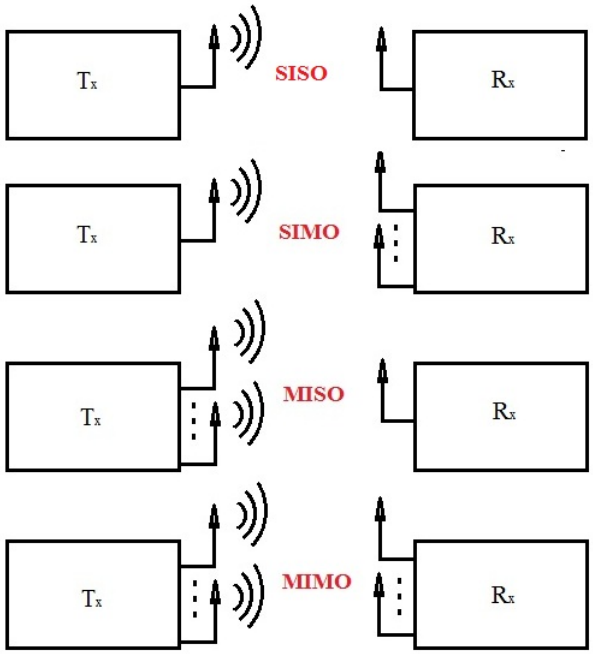

FIGURE 1.2 – Différentes configurations d'antennes

1.4 Gain de diversité :

Lorsque le signal sortant du canal est fortement affecté on dit qu'il a subit un évanouissement.Donc pour résoudre ce problème l'envoi de plusieurs réplique du même signal sur différentes branches c'est ce qu'on appelle «la diversité».

- Diversité temporelle :
  Consiste à envoyer le même signal à des instants différents séparés d'au moins le temps de cohérence du canal pour assurer leur décorrélation. Cette technique de diversité est intéressante pour les canaux ergodique.
  Dans ce cas les répliques du signal transmis sont fourni à travers le temps par une combinaison de codage de canaux et stratégies d'entrelacement de temps.
  Pour que cette forme de diversité soit effective, il faut que le canal fournit assez de variation en temps. On l'applique dans le cas ou la cohérence en temps du canal est inferieur au temps de l'entrelacement des symboles désiré.
  Dans ce cas, on est sur que les symboles sont indépendants les uns aux autres et rend une nouvelle réplique du symbole originale.
- Diversité fréquentielle :
  consiste à envoyer le même signal sur des fréquences différentes séparées d'au moins la bande de cohérence du canal. Cette technique de diversité est utilisée dans les systèmes OFDM.
  Elle fournit des répliques du signal original sur le domaine en fréquence.Cela garantit que les différentes parties du spectre soient affectées par le même évanouissement.
- Diversité spatiale :
  La diversité d'espace en émission : consiste à envoyer le même signal sur des antennes différentes séparées d'au moins dix fois la longueur d'onde.
  A la réception, cette diversité est perçue comme de la diversité temporelle. On l'appelle aussi « diversité d'antennes », c'est une méthode efficace pour combattre l'évanouissement multi trajets, car elle envoie plusieurs répliques du même signal sur les différentes antennes du récepteur. Cette technique est appliquée lorsque la distance entre les antennes est plus grande que la distance cohérente afin d'assurer l'indépendance entre les signaux reçu par les différentes antennes .
  Ce type diversité est obtenue grâce à l'application de méthode de codage espace-temps.
  Diversité d'espace en réception : C'est la réception du même signal sur plusieurs antennes différentes séparées d'au moins dix fois la longueur d'onde. Cette diversité est obtenue grâce à l'utilisation d'un systèmes à plusieurs antennes en émission et en réception.

1.4.1 Modèles des canaux d'évanouissements :

1.4.1.1 Canal de Rayleigh :

Le signal reçu est perturbé par l'évanouissement dû au trajets multiples ainsi que AWGN (Voir figure 1.3). Basé sur la bande passante de cohérence du canal et la bande passante du signal, deux modèles existent[6] :
Systèmes à bande étroite (flat frequency) : Dans ce type de systèmes, les signaux transmis occupent généralement une bande passante inférieure à la bande passante de

cohérence du canal ; Ce type est également appelé « système à fréquence non sélectif »car toutes les fréquences du signal s'évanouissent de manière égale.
Systèmes à large bande : Dans ce type de systèmes, la bande passante du signal transmis est plus grande que la bande passante de cohérence du canal. Par conséquent, Les composantes spectrales du signal transmis avec une séparation de fréquence plus grande que la bande passante de cohérence s'évanouissent indépendamment.
Dans le cas de l'évanouissement non-sélective les signaux reçu dans une bande de base complexe est représenté comme suit : $r(t) = h(t)x(t) + n(t)$
Où, $x(t)$ et $r(t)$ sont les signaux de bande de base complexes transmis et reçu, respectivement, et h (t) est l'information dur l'état du canal (CSI)[6].

1.4.1.2 Canal de Rice :

Si la moyenne des évanouissements n'est pas nulle. Les signaux passent par des chemins latéraux et possèdent un chemin directe (LOS stable).
Donc, on a une composante principal et des composantes de phase aléatoire.
Voir figure 1.3 .

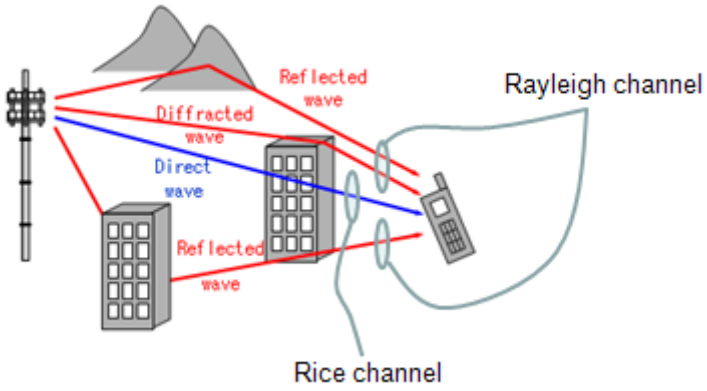

FIGURE 1.3 – Trajets des signaux

1.4.1.3 Propagation multi-trajets :

L'environnement du récepteur cause des réflexions des ondes avec atténuation d'amplitude et de phase ; ces ondes réfléchis sont appelés ondes à trajets multiple.
La combinaison de ces ondes à l'entrée du récepteur peut être soit constructive ou destructive selon la phase du signal .La fluctuation en amplitude est appelée une décoloration du signal dû à la variation en temps des trajets.

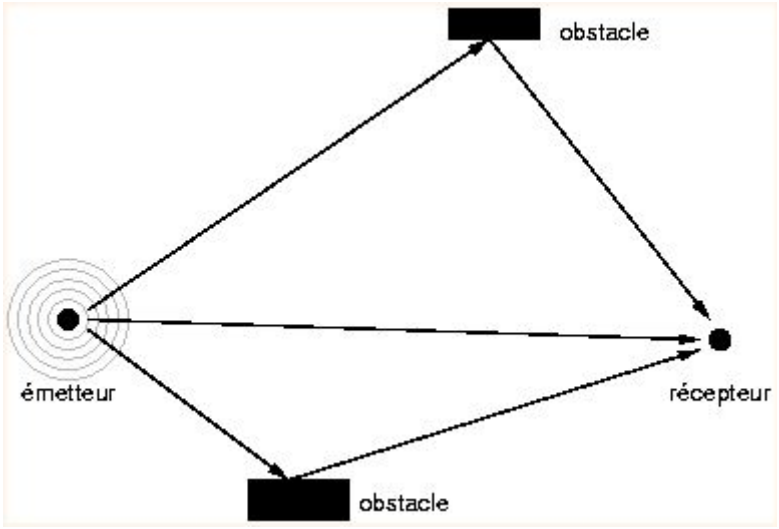

FIGURE 1.4 – trajets multiples des signaux

1.4.1.4 Doppler shift :

Ce phénomène est le résultat du déplacement et la mobilité du récepteur par rapport à l'émetteur. Il introduit un décalage en fréquence du signal reçu et dépends essentiellement de[2] :

- La position du mobile.
- La vitesse du récepteur v.

Soit :
λ :la longueur d'onde.
f :la fréquence porteuse.
f' :la fréquence captée par le récepteur

$$f' = f - \frac{v}{\lambda} \tag{1.2}$$

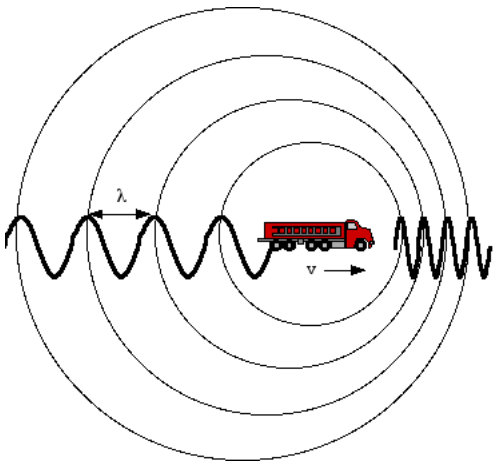

FIGURE 1.5 – effet doppler

1.4.2 Evanouissement plat :

C'est le cas où les différences de chemins sont faibles (par rapport à la durée des impulsions). La réponse en fréquence du canal est plate, ce qui veut dire que toutes les fréquences subit la même atténuation et ne déforme ni la phase ni l'amplitude du signal[7].
$\Delta t < T_s$
Si m(t) est le signal transmis alors m'(t) est le signal en sortie du canal tel que :

$$m'(t) = [h_1(t) + h_2(t) + ... + h_{N-1}(t) + h_N(t)] \otimes m(t)$$

propriétés :

1. Même un faible déplacement (du mobile ou d'un élément extérieur, peut entraîner une réponse assez différente).
2. La cohérence spatiale et la cohérence temporelle sont faibles.
3. Si la pseudo-stationnarité (stationnaire pendant la durée d'un symbole, ou d'une trame) n'est pas vérifiée, on ne peut pas faire grand chose ...
4. La réponse du canal est simplement un coefficient complexe mais variable au cours du temps.

1.5 Système de communication MIMO

Dans une chaine de transmission MIMO, l'envoi du flux d'information est réalisé à travers N antennes à l'émission et M antennes en réception. le flux d'information venant de la source est converti par l'émetteur, passant par le canal pour enfin arriver au récepteur qui effectuera une opération inverse afin de restituer le message au destinataire.
Un système est dit cohérent si le récepteur connait les coefficients du canal. Dans les systèmes non cohérents, une estimation des coefficients du canal est nécessaire.
Dans ce travail, un système MIMO 2 X 2 (deux antennes en émission et en réception) non cohérent est utilisé, avec un estimateur de canal à séquence d'apprentissage.

1.6 Codage spatio-temporel :

1.6.1 Définitions :

L'idée de base des CST repose sur la transmission adéquate de l'information parmi les différentes antennes et dans le temps. Il s'agira donc d'associer la dimension temporelle du codage de canal "classique" avec la dimension spatiale (dans le sens géographique du terme) des multi-antennes. Ainsi, le raisonnement ce fait sur 2 dimensions [5].
le codage est spatial, temporel pour introduire une corrélation entre les signaux transmis.
Soit X la matrice de transmission de n x p taille tel que n est le nombre d'antennes de transmissions et p le nombre de périodes de transmission d'un bloc de symboles codés.
Si on prends une constellation de 2^m de points. Pour chaque opération de codage un bloc de k.m bits d'informations est choisi tel que k signal modulé $(x_1, x_2, x_3, x_4, \ldots x_K)$ et chaque groupe de m bits selecte un signal de constellation.

Les k signaux sont codés par le codeur STBC pour générer n signaux parallèles dans p périodes d'où la matrice X, ces séquences sont transmis à travers n antennes simultanément sur p périodes.

$$X_{n,P} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{1.3}$$

1.6.2 Codage spatio-temporel en bloc :

le STBC à comme entrée k symboles donc chaque antenne d'émission transmet p symboles pour chaque block de K[1].

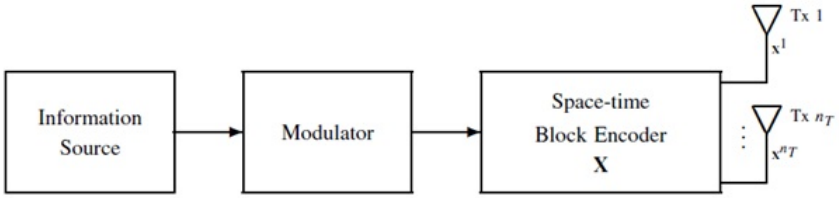

FIGURE 1.6 – codeur spatio-temprel

R est le rendement (rate) du STBC défini comme le rapport entre K et p.

$$R = \frac{K}{P}$$

La capacité spectrale du stbc est :

$$\eta = \frac{r_b}{B} = \frac{r_s.m.R}{r_s} = \frac{K.m}{p}\ bits/s/Hz$$

r_b : $débit\ binaire(Bit\ rate)$.
r_s : $débit\ de\ symbole(symbole\ rate)$.
B : $la\ bande\ passante$.

Les entrées du STBC sont des combinaisons linéaires des k symboles modulés et leurs conjugués. Pour avoir une diversité totale de n, la matrice X est construite de tel manière à avoir un système orthogonale.

$$X.X^H = C(|x_1|^2 + |x_2|^2 + \ldots + |x_k|^2).I_n$$

tel que :
C : Constante.
I_n : matrice identité de n x n.

la i ^{e} ligne de X représente les symboles transmis par la i ^{e}antenne dans p périodes consécutives.
La j ^{e} colonne de X représente les symboles transmis simultanément par n antennes dans le temps j .

la j [e] colonne est prise comme un space time symbole transmis au temps j.
les éléments de la matrice X, $x_{i,j}$ $\{i = 1, \ldots, n\ , j = 1, \ldots, p\}$ représentent les signaux transmis par l'antenne i au temps j.
Le rendement d'un code d'un bloc spatio-temporel à une diversité totale est $R \leq 1$. On note X_n la matrice de transmission et est appelé le code spatio-temporel de taille n. Rappelons que le système est orthogonale cela veut dire que dans chaque bloc la séquence des signaux de chaque 2 antennes sont orthogonaux. Cela permettra d'attendre la diversité totale ainsi que le découplage facile par le récepteur.
Selon le type de constellation on peut classifier les codes spatio-temporel en codes à signaux réels et codes à signaux complexes.

1.6.2.1 STBC pour une constellation réelle :

Si on prend la matrice X_n carrée pour une constellation arbitraire de M-ASK, le code spatio-temporel à matrice de transmission carrée n x n existe si est seulement si $n = 2, 4, 8$ ces codes sont de rendement R=1 est de diversité totale de n [1].
Pour un bloc de k symboles modulés, le nombre d'antenne de transmission n est égal au nombre de période de transmission p qui sont égaux à la taille du bloc message k. D'ou pas besoin d'une expansion de la bande passante est le code peut atteindre un rendement de 1 [1].

$$X_2 = \begin{bmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{bmatrix} \tag{1.4}$$

$$X_4 = \begin{bmatrix} x_1 & -x_2 & -x_3 & -x_4 \\ x_2 & x_1 & x_4 & -x_3 \\ x_3 & -x_4 & x_1 & x_2 \\ x_4 & x_3 & -x_2 & x_1 \end{bmatrix} \tag{1.5}$$

pour n antennes la valeur minimum de période de transmission pour atteindre un rendement totale est : min(2^{4c+d}) tel que : $c, d\ |0 \leq c, 0 \leq d \leq 4\ et\ 8c + 2d \geq n$
Pour $n \leq 8$:
n = 2, p= 2
n = 3, p= 4
n = 4, p= 4
n = 5, p= 8
n = 6, p= 8
n = 7, p= 8
n = 8, p= 8
D'où les matrices X_3, X_5, X_6, X_7 sont de diversité et rendement totale. Comme l'entrée de codeur spatio-temporel est un bloc de 8 symboles d'une constellation réel on retrouve les 8 symboles dans les périodes de transmissions donc pas besoin d'une expansion de la bande passante.

1.6.2.2 STBC pour une constellation complexe :

Le code d'Alamouti fournit un rendement totale juste pour une matrice carrée de 2X2 [1].

$$X_2^c = \begin{bmatrix} x_1 & -x_2^* \\ x_2 & x_1^* \end{bmatrix} \tag{1.6}$$

Pour un nombre plus grand que deux d'antennes, le but c'est de construire une matrice à rendement maximum avec une complexité faible du codage, de même le nombre de période doit être minimiser pour minimiser le temps de codage.
On peut trouver des matrices X_3^c, X_4^c à rendement $1/2$. Comme on peut trouver X_3^h, X_4^h à rendement $3/4$ [1].

$$X_3^c = \begin{bmatrix} x_1 & -x_2 & -x_3 & -x_4 & x_1^* & -x_2^* & -x_3^* & -x_4^* \\ x_2 & x_1 & x_4 & -x_3 & x_2^* & x_1^* & x_4^* & -x_3^* \\ x_3 & -x_4 & x_1 & x_2 & x_3^* & -x_4^* & x_1^* & x_2^* \end{bmatrix} \tag{1.7}$$

$$X_4^c = \begin{bmatrix} x_1 & -x_2 & -x_3 & -x_4 & x_1^* & -x_2^* & -x_3^* & -x_4^* \\ x_2 & x_1 & x_4 & -x_3 & x_2^* & x_1^* & x_4^* & -x_3^* \\ x_3 & -x_4 & x_1 & x_2 & x_3^* & -x_4^* & x_1^* & x_2^* \\ x_4 & x_3 & -x_2 & x_1 & x_4^* & x_3^* & -x_2^* & x_1^* \end{bmatrix} \tag{1.8}$$

$$X_3^h = \begin{bmatrix} x_1 & -x_2^* & \frac{x_3^*}{\sqrt{2}} & \frac{x_3^*}{\sqrt{2}} \\ x_2 & x_1^* & \frac{x_3^*}{\sqrt{2}} & \frac{-x_3^*}{\sqrt{2}} \\ \frac{x_3}{\sqrt{2}} & \frac{(-x_1-x_1^*+x_2-x_2^*)}{\sqrt{2}} & \frac{x_1-x_1^*+x_2+x_2^*}{\sqrt{2}} & \end{bmatrix} \tag{1.9}$$

$$X_4^h = \begin{bmatrix} x_1 & -x_2^* & \frac{x_3^*}{\sqrt{2}} & \frac{x_3^*}{\sqrt{2}} \\ x_2 & x_1^* & \frac{x_3^*}{\sqrt{2}} & \frac{-x_3^*}{\sqrt{2}} \\ \frac{x_3}{\sqrt{2}} & \frac{(-x_1-x_1^*+x_2-x_2^*)}{\sqrt{2}} & \frac{x_1-x_1^*+x_2+x_2^*}{\sqrt{2}} & \\ \frac{x_3}{\sqrt{2}} & \frac{-x_3}{\sqrt{2}} & \frac{(x_1-x_1^*-x_2-x_2^*)}{\sqrt{2}} & \frac{(-x_1-x_1^*-x_2+x_2^*)}{\sqrt{2}} \end{bmatrix} \tag{1.10}$$

1.7 Emission :

L'émission se résume sur 3 étapes. La première est la source d'information qui envoie un signal numérique, suivie d'une modulation par (un modulateur BPSK ou QAM) et enfin codé par le codeur d'Alamouti (voir figure 1.7)

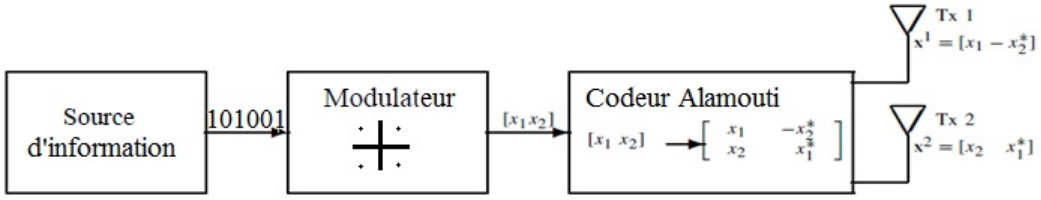

FIGURE 1.7 – Chaine d'émission

1.7.1 Modulation :

1.7.1.1 Modulation bpsk :

Pour cette modulation un seul bit est traité (Voir : code vhdl en Annexe B) tel que :

Symboles	Modulation
1	1
0	-1

Table 1.1 – Tableau de la modulation BPSK

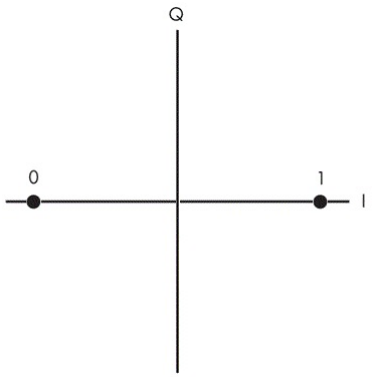

Figure 1.8 – Constellation BPSK

1.7.1.2 Modulation QAM :

Les symboles (informations) sont de taille 2 bits. Les signaux générés par le modulateur présentent deux composantes (Partie Réelle, imaginaire) comme suit :

Symboles	Modulation
10	$1-i$
00	$-1-i$
01	$-1+i$
11	$1+i$

Table 1.2 – Tableau de la modulation QAM

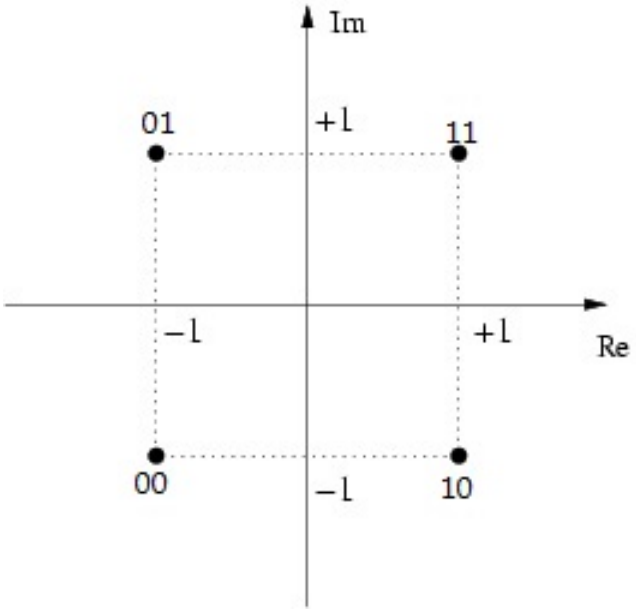

FIGURE 1.9 – Constellation QAM

Voir :code vhdl en Annexe B.

1.7.2 Codage spatio-temporel d'Alamouti :

Le code d'alamouti est le premier code qui atteint une diversité totale pour 2 antennes. Un M-array est utilisé pour la modulation. Chaque groupe de m bits d'information est modulé, tel que $m = log_2 M$. Le codeur prend un bloc de 2 symboles modulés x_1 et x_2 pour chaque opération de codage et les transmet aux antennes de transmission selon une matrice de code[1].

$$X = \begin{bmatrix} x_1 & -x_2^* \\ x_2 & x_1^* \end{bmatrix} \tag{1.11}$$

Les colonnes de la matrice representent les periodes de transmission t,$t+1$. Les lignes representent les antennes. Durant la première période, la première antenne transmet le symbole x_1 et la deuxieme transmet x_2. Durant la periode qui suit la première transmet $-x_2^*$ et la deuxieme transmet x_1^* [1]. La matrice du code possède la propriété d'orthogonalité suivante :

$$X.X^H = \begin{bmatrix} |x_1|^2 + |x_2|^2 & 0 \\ 0 & |x_1|^2 + |x_2|^2 \end{bmatrix} = (|x_1|^2 + |x_2|^2) I_2 \tag{1.12}$$

Le rendement du code d'alamouti pour 2 antennes est $R = 1$.

la capacité spectrale pour BPSK $m = 1$ est de : $\eta = 1$ et pour le QAM $m = 2$ est de $\eta = 2$. Pour permettre au recpeteur d'avoir le CSI, le codeur envoie dans un premier temps une séquence d'identification dont le récepteur connait aussi.

1.7.2.1 Performance du code d'Alamouti

Soit B la matrice différence de mots de code[9] :

$$B(X,\widehat{X}) = \begin{bmatrix} x_1 - \widehat{x}_1 & -x_2^* + \widehat{x}_2^* \\ x_2 - \widehat{x}_2 & x_1^* - \widehat{x}_1^* \end{bmatrix} \tag{1.13}$$

Soit X la matrice distance de mot de code (CDM) :

$$A(X,\widehat{X}) = B(X,\widehat{X})B^H(X,\widehat{X}) \begin{bmatrix} |x_1 - \widehat{x}_1|^2 + |x_2 - \widehat{x}_2|^2 & 0 \\ 0 & |x_1 - \widehat{x}_1|^2 + |x_2 - \widehat{x}_2|^2 \end{bmatrix} \tag{1.14}$$

Le minimum de distance entre chaque 2 codes transmit reste le même que celui d'un système non codé. Donc, le code d'Alamouti ne génère pas de gain de codage (CGD).

$$G_c = \frac{(\lambda_1\lambda_2)^{1/2}}{d_E^2} = 1$$

(1.15)

Tel que les valeurs propres de la matrice distance du mot de code est égale à la distance euclidienne.

$$d_E^2(X,\widehat{X}) = |x_1 - \widehat{x}_1|^2 + |x_1 - \widehat{x}_1|^2 + |x_2 - \widehat{x}_2|^2$$

(1.16)

La performance du code d'Alamouti, sur un canal de Rayleigh quasi-statique, à évanouissement est illustré sur la figure 1.10. Le système étudié est constitué d'une seule antenne de réception et utilise une modulation QPSK. On trace sur la figure 1.10, la probabilité d'erreur par symbole, en fonction du SNR. Comme nous pouvons le constater, la performance du code d'Alamouti avec deux antennes d'émission est bien meilleure que celle d'un système à une antenne d'émission. Pour un taux d'erreur par symbole de 10^{-3} le code d'Alamouti offre un gain de 11 dB. Ce qui est plus important de constater encore, c'est que plus le SNR augmente, plus l'écart de performance entre les deux systèmes augmente. En fait le taux d'erreur diminue de façon inversement proportionnel au carré du SNR[4].

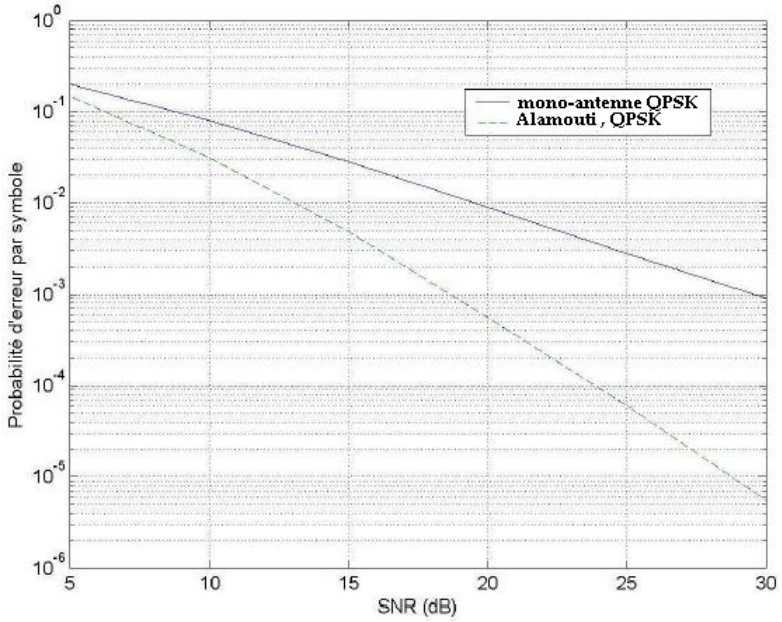

FIGURE 1.10 – performance du code d'alamouti

1.8 Réception

Aprés passage des signaux par le canal, la séquence d'apprentissage du canal pour avoir le CSI passe à travers l'estimateur du canal pour tirer la matrice du canal qui va etre envoyer apres vers le decodeur qui recoit les signaux sortant du canal.

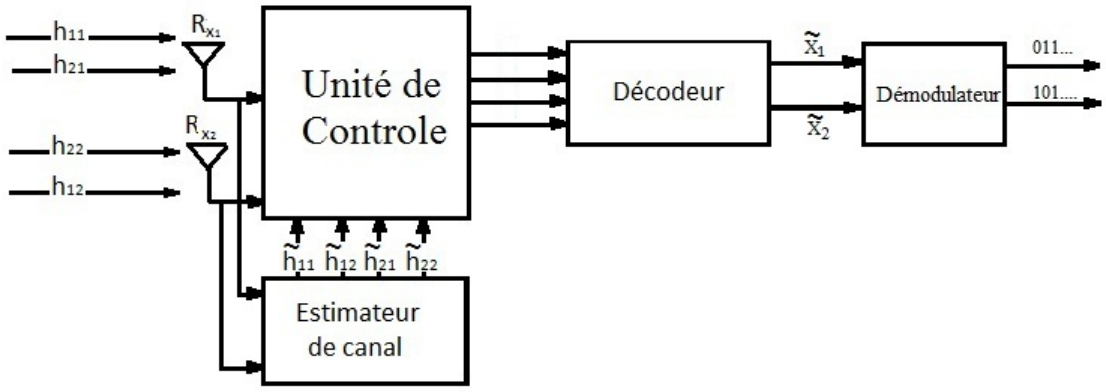

FIGURE 1.11 – Chaine de reception

1.8.1 Décodage

Les signaux reçu à la réception sont selon l'équation suivante :

$$y = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_1 & -x_2^* \\ x_2 & x_1^* \end{bmatrix} + \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \tag{1.17}$$

$$y = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \tag{1.18}$$

$$r_{11} = h_{11}x_1 + h_{12}x_2 + e_{11} \tag{1.19}$$
$$r_{12} = -h_{11}x_2^* + h_{12}x_1^* + e_{12} \tag{1.20}$$
$$r_{21} = h_{21}x_1 + h_{22}x_2 + e_{21} \tag{1.21}$$
$$r_{22} = -h_{21}x_2^* + h_{22}x_1^* + e_{22} \tag{1.22}$$

$$\begin{cases} \widetilde{x_1} & = h_{11}^* r_{11} + h_{12} r_{12}^* + h_{21}^* r_{21} + h_{22} r_{22}^* \\ \widetilde{x_2} & = h_{12}^* r_{11} - h_{11} r_{12}^* + h_{22}^* r_{21} - h_{21} r_{22}^* \end{cases} \tag{1.23}$$

En developpant la multiplication entre complexe et en séparant la partie réelle de la partie imaginaire on trouve :

$$\begin{aligned} \widetilde{x}_{1re} = {} & Re(h_{11})Re(r_{11}) + Im(h_{11})Im(r_{11}) + Re(h_{12})Re(r_{12}) + Im(h_{12})Im(r_{12}) \\ & + Re(h_{21})Re(r_{21}) + Im(h_{21})Im(r_{21}) + Re(h_{22})Re(r_{22}) + Im(h_{22})Im(r_{22}) \end{aligned} \tag{1.24}$$

$$\begin{aligned} \widetilde{x}_{1im} = {} & Re(h_{11})Im(r_{11}) - Im(h_{11})Re(r_{11}) - Re(h_{12})Im(r_{12}) + Im(h_{12})Re(r_{12}) \\ & + Re(h_{21})Im(r_{21}) - Im(h_{21})Re(r_{21}) - Re(h_{22})Im(r_{22}) + Im(h_{22})Re(r_{22}) \end{aligned} \tag{1.25}$$

$$\begin{aligned} \widetilde{x}_{2re} = {} & Re(h_{12})Re(r_{11}) + Im(h_{12})Im(r_{11}) - Re(h_{11})Re(r_{12}) - Im(h_{11})Im(r_{12}) \\ & + Re(h_{22})Re(r_{21}) + Im(h_{22})Im(r_{21}) - Re(h_{21})Re(r_{22}) - Im(h_{21})Im(r_{22}) \end{aligned} \tag{1.26}$$

$$\begin{aligned} \widetilde{x}_{2im} = {} & Re(h_{12})Im(r_{11}) - Im(h_{12})Re(r_{11}) + Re(h_{11})Im(r_{12}) - Im(h_{11})Re(r_{12}) \\ & + Re(h_{22})Im(r_{21}) - Im(h_{22})Re(r_{21}) + Re(h_{21})Im(r_{22}) - Im(h_{21})Re(r_{22}) \end{aligned} \tag{1.27}$$

1.8.2 Estimateur du canal :

La matrice X sortant du codeur passe par le canal H pour arriver au recepteur.
Si le recepteur peut extraire les coeffecients du canal d'où la matrice du canal, elle sera utilisé par le décodeur comme l'information sur l'etat du canal (channel state information (CSI)).
En partant de la théorie, on peut déterminer les coefficients H_{ij} de la manière suivante :
Dès que l'estimateur reçoit la séquence d'identification et sachant qu'il connait les signaux d'origines, il peut extraire la matrice H comme suit :

$$\hat{H} = Y_c X_c^H (X_c X_c^H)^{-1} \tag{1.28}$$

Tel que :
X_c est la matrice connue envoyé par l'émetteur.
Y_c est la matrice reçu.
$(.)^H$ est la matrice Hermitienne conjuguée.
l'équation 1.28 dépends de la séquence d'apprentissage qui est faite de telle manière qu'elle satisfait l'équation suivante :

$$X_c X_c^H = \rho^2 I \tag{1.29}$$

Sachant que aprés passage de la séquence par le canal on a :

$$Y_c = HX_c + E \tag{1.30}$$

Et en substituant l'équation 1.30 dans 1.28 on obtient :

$$\begin{aligned} \hat{H} &= (HX_c + E)X_c^H(X_c X_c^H)^{-1} \\ &= HX_c X_c^H(X_c X_c^H)^{-1} + X_c^H(X_c X_c^H)^{-1}E \\ &= H + EX_c^H \rho^2 I \\ &= H + terme\ d'erreur \end{aligned} \tag{1.31}$$

1.9 Conclusion

Nous avons présenté dans ce chapitre les fondements théoriques des techniques de codage / décodage spatio-temporels des systèmes MIMO.
Il est montré notamment que la méthode d'Alamouti présente d'excellentes performances en terme de décodage (simplicité, diversité (maximale) et de rendement (unitaire).
Ces performances feront l'objet d'une étude en simulation dans le chapitre suivant.

Chapitre 2

Simulation

2.1 introduction

Dans cette partie, le codeur/ décodeur d'Alamouti sera validé par une simulation de haut niveau à l'aide de l'outil Matlab(Simulink). Le schéma bloc utilisé est celui traité dans le chapitre précédent. Des blocs pour le décodage et l'estimation seront ajoutés selon la partie théorique déja traité.

2.2 Simulation Matlab(Simulink) :

Nous avons décrit en MATLAB les principaux blocs intervenant dans la chaine de décodage d'Alamouti.

2.2.1 Estimateur de canal :

Le décodage d'Alamouti exige la connaissance en temps réel des caractéristiques du canal, l'estimateur a été développé en MATLAB pour satisfaire cette exigence(chan_est.m).
Le code source pour l'estimateur du canal peut être trouvé dans l'annexe A.2

2.2.2 Décodeur d'Alamouti :

Le décodeur d'Alamouti est composé de 2 circuits, le combineur et le détecteur de symbole, le code Matlab (decodeur.m) décrit le fonctionnement du circuit. Le code source du décodeur peut être trouvé dans l'annexe A.3.

2.2.3 Simulateur de canal :

Pour le besoin de simulation, on a développé un canal de Rayleigh MIMO à base de composants SISO Rayleigh (voir : figure 2.1) et le bloc AWGN.

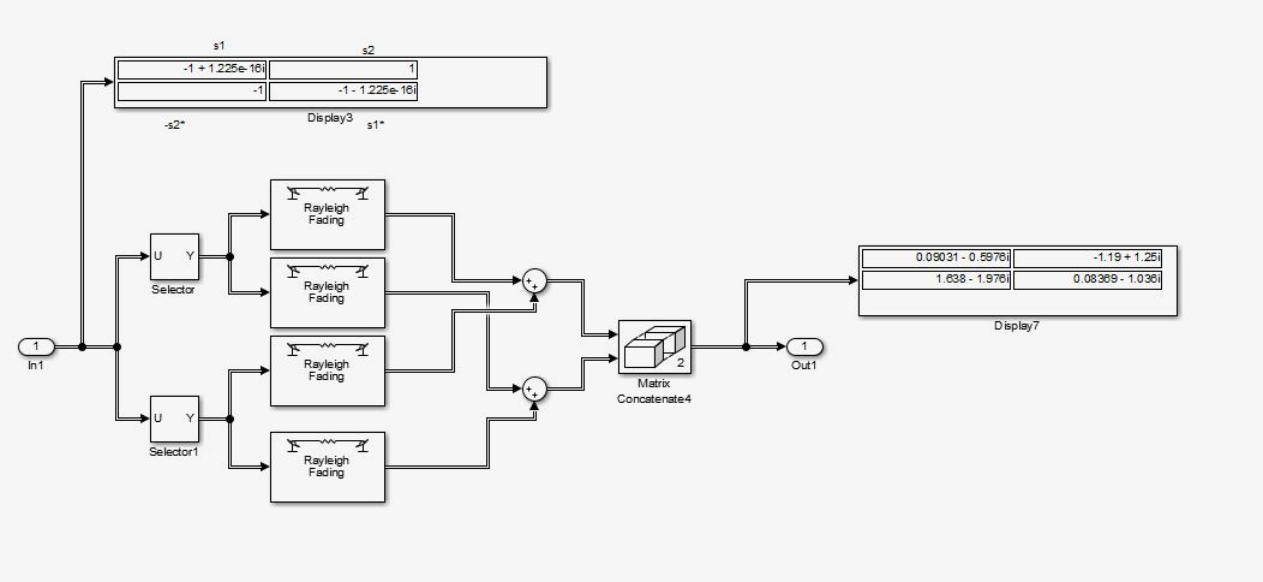

FIGURE 2.1 – canal de rayleighrayleigh

2.2.4 Résultats de la simulation MATLAB(Simulink) :

Le test est fait pour la modulation BPSK. On envoie des signaux aléatoire (en utilisant le générateur Binaire de Bernouli) et on remarque bien que le décodeur arrive à extraire les signaux source.
La figure qui suit illustre le résultat de la simulation.

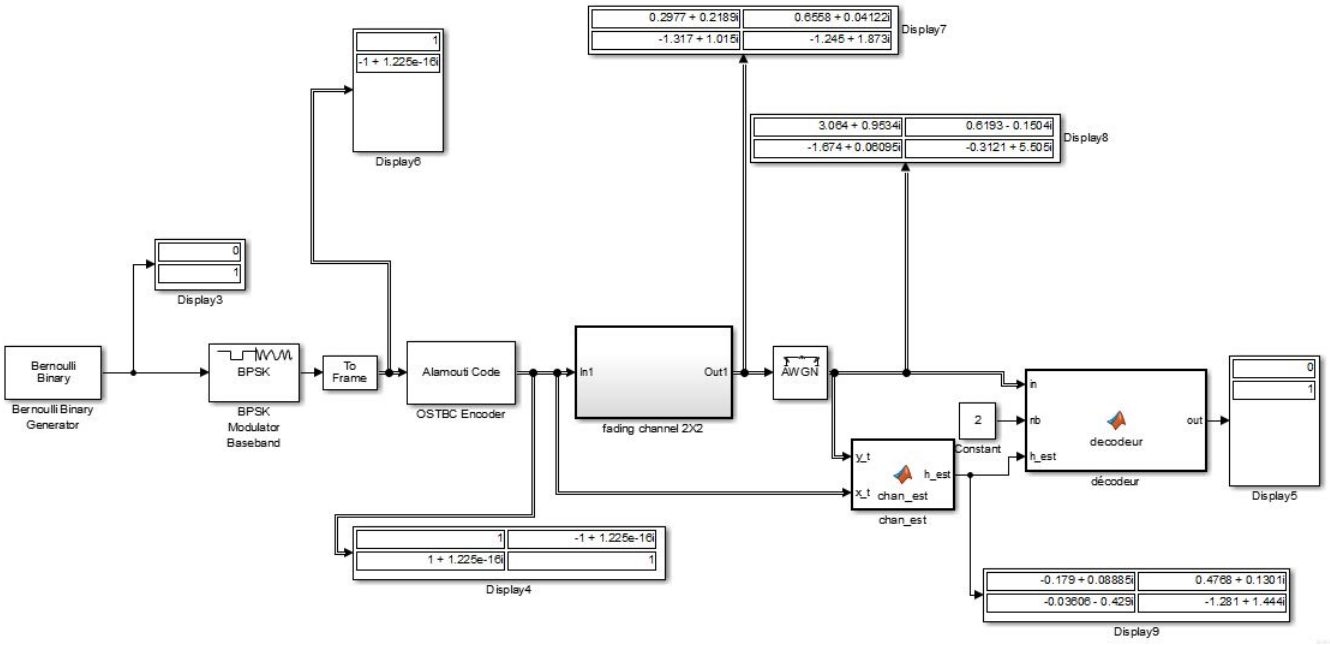

FIGURE 2.2 – schema simulink

La figure 2.2 montre un seul résultat, comme le décodeur utilise une décision soft et une autre Hard grâce au détecteur de seuil, le résultat final est soit un bit nul ou un tel que le bit 1 est pour le cas positif et le zéro pour le cas négatif, ce qui assure que le bruit n'affecte pas les sorties du décodeur.

2.3 Conclusion :

les résultats obtenus lors des simulations ont confirmé le bon fonctionnement du décodeur d'Alamouti.
L'implémentation sur la carte FPGA sera traité dans le prochain chapitre.

Chapitre 3

implémentation

3.1 introduction

Après vérification des codes par simulation, ce chapitre traitera la partie implémentation sur la carte FPGA ML501. les résultats de l'implémentation seront représentés.

3.2 Modélisation HDL :

Cette modélisation a été réalisé à l'aide de l'outil ModelSim de Mentor Graphics, qui fournit un environnement complet de simulation et débogage pour les designs matériels. Il supporte plusieurs langages de description[3]tel que :

- VHDL
- Verilog
- Verilog 2001
- SystemVerilog
- PSL
- SystemC

3.2.1 Codeur d'Alamouti :

Ce codeur suit la description donnée dans la section 1.7, avec un modulateur détaché.Il prends en entrée deux vecteurs (std_logic_vector) de 16 bits, 8 bits partie entière et 8 pour la partie fractionnaire. Donne comme sortie deux matrice (2×2) une qui contient les parties réelles et l'autre qui contient les parties imaginaires. Dans un premier lieu le codeur envoie une séquence d'apprentissage pour permettre l'estimation du canal. On a choisi les signaux $s_1 = 1 + i$ et $s_2 = 1 + i$ que l'estimateur connait. Après cela, le temps de la séquence s'écoule le codeur commence à recevoir en entrée les sorties des modulateurs.
Le code vhdl peut être trouvé en annexe B.2.

3.2.2 estimateur :

L'estimateur reçoit la séquence d'apprentissage qui passent par le canal. Sachant que X_c est : $s_1 = 1 + i$ et $s_2 = 1 + i$

$$X_c = \begin{bmatrix} 1+i & -1+i \\ 1+i & 1-i \end{bmatrix} \tag{3.1}$$

On calcule la partie constante $X_c^H(X_cX_c^H)^{-1}$ qui sera prédefinie dans le code pour simplifier les opérations, on trouve :

$$X_c^H(X_cX_c^H)^{-1} = \begin{bmatrix} 1-i & 1-i \\ 1-i & -1+i \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} + i \begin{bmatrix} -1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.2}$$

D'où :

$$H = \begin{bmatrix} r_{11c} & r_{12c} \\ r_{21c} & r_{22c} \end{bmatrix} \times \begin{bmatrix} 1-i1 & 1-i1 \\ 1-i1 & -1+i1 \end{bmatrix} \tag{3.3}$$

L'estimateur s'est simplifié pour devenir juste une multiplication entre complexe.
le Code VHDL peut etre retrouvé en annexe B.5.

3.2.3 Décodeur d'Alamouti :

Le bloc le plus important de ce décodeur est l'unité de control qui reçoit et affecte les valeurs nécessaire pour chaque bloc selon les équations (1.24),(1.25),(1.26),(1.27), le nombre de cycles d'horloge nécessaire pour le calcul est de 8 cycles.
En vue de simplifier la multiplication et diminuer le nombre de cycles d'où le temps de calcul, on modifie l'unité de control qui commande et affecte 8 opérandes sur 4 états (cycles) les valeurs selon le tableau 3.1 qui suit :

Opérande	1ercycle	2^{e}cycle	3^{e}cycle	4^{e}cycle
a	$Re(h_{11})$	$Im(h_{11})$	$Re(h_{12})$	$Im(h_{12})$
b	$Re(r_{11})$	$Im(r_{11})$	$Re(r_{12})$	$Im(r_{12})$
c	$Im(r_{11})$	$Re(r_{11})$	$Im(r_{12})$	$Re(r_{12})$
d	$Re(h_{12})$	$Im(h_{12})$	$Re(h_{11})$	$Im(h_{11})$
a_2	$Re(h_{21})$	$Im(h_{21})$	$Re(h_{22})$	$Im(h_{22})$
b_2	$Re(r_{21})$	$Im(r_{21})$	$Re(r_{22})$	$Im(r_{22})$
c_2	$Im(r_{21})$	$Re(r_{21})$	$Im(r_{22})$	$Re(r_{22})$
d_2	$Re(h_{22})$	$Im(h_{22})$	$Re(h_{21})$	$Im(h_{21})$

TABLE 3.1 – tableau d'affectation des opérandes au décodeur

$$\tilde{x}_{1\_re} = \sum_{n=1}^{4}(a_ib_i + a_{2i}b_{2i}), \quad \tilde{x}_{1\_im} = \sum_{n=1}^{4}(a_ic_i + a_{2i}c_{2i})$$

$$\tilde{x}_{2\_re} = \sum_{n=1}^{4}(b_id_i + b_{2i}d_{2i}), \quad \tilde{x}_{2\_im} = \sum_{n=1}^{4}(d_ic_i + d_{2i}c_{2i})$$

Voici le schéma de la partie décodage.

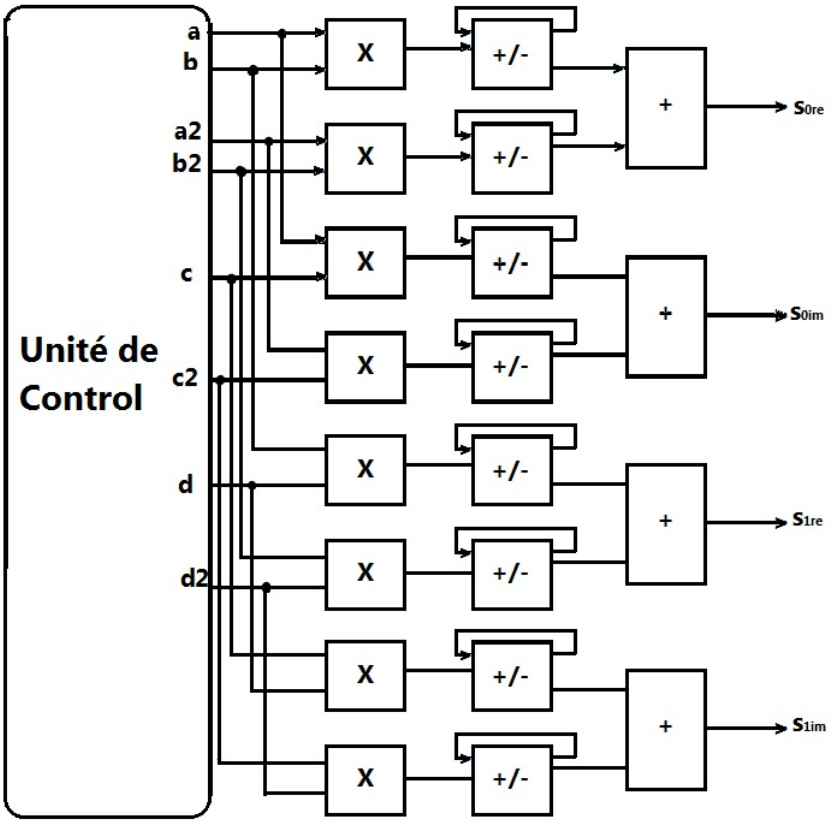

FIGURE 3.1 – décodeur

Le code vhdl du décodeur peut être trouvé en annexe B 2.4.

3.3 Simulateur du canal :

Pour tester notre décodeur, on a établit un bloc VHDL qui simule le canal de transmission. La fonction du canal étant la multiplication des matrices d'émission et celle du canal H.

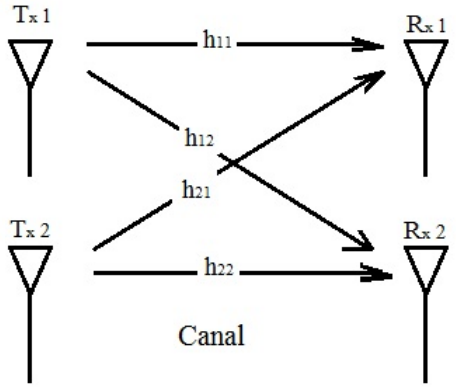

FIGURE 3.2 – Canal

la matrice du canal considérée est :

$$H = \begin{bmatrix} 128 + i48 & 192 + i64 \\ 64 + i192 & 48 + i128 \end{bmatrix} \tag{3.4}$$

La mutliplication entre complexe donne comme resultat :

$$\widetilde{r}_{0re} = Re(h_{11})Re(x_{11}) - Im(h_{11})Im(x_{11}) + Re(h_{12})Re(x_{21}) - Im(h_{12})Im(x_{21}) \tag{3.5}$$

$$\widetilde{r}_{1re} = Re(h_{11})Re(x_{12}) - Im(h_{11})Im(x_{11}) + Re(h_{12})Re(x_{22}) - Im(h_{12})Im(x_{22}) \tag{3.6}$$

$$\widetilde{r}_{2re} = Re(h_{21})Re(x_{11}) - Im(h_{21})Im(x_{11}) + Re(h_{22})Re(x_{21}) - Im(h_{22})Im(x_{21}) \tag{3.7}$$

$$\widetilde{r}_{3re} = Re(h_{21})Re(x_{12}) - Im(h_{21})Im(x_{12}) + Re(h_{22})Re(x_{22}) - Im(h_{22})Im(x_{22}) \tag{3.8}$$

$$\widetilde{r}_{0im} = Re(h_{11})Im(x_{11}) + Im(h_{11})Re(x_{11}) + Re(h_{12})Im(x_{21}) + Im(h_{12})Im(x_{21}) \tag{3.9}$$

$$\widetilde{r}_{1im} = Re(h_{11})Im(x_{12}) + Im(h_{11})Re(x_{12}) + Re(h_{12})Im(x_{22}) + Im(h_{12})Re(x_{22}) \tag{3.10}$$

$$\widetilde{r}_{2im} = Re(h_{21})Im(x_{11}) + Im(h_{21})Re(x_{11}) + Re(h_{22})Im(x_{21}) + Im(h_{22})Re(x_{21}) \tag{3.11}$$

$$\widetilde{r}_{3im} = Re(h_{21})Im(x_{12}) + Im(h_{21})Re(x_{12}) + Re(h_{22})Im(x_{22}) + Im(h_{22})Re(x_{22}) \tag{3.12}$$

Les valeurs de chaques opérandes sont affectées selon le tableau 3.4.

$$\tilde{r}_{11\_re} = \sum_{n=1}^{4} a_i b_i, \quad \tilde{r}_{11\_im} = \sum_{n=1}^{4} a_i d_i$$

$$\tilde{r}_{12\_re} = \sum_{n=1}^{4} a2_i b2_i, \quad \tilde{r}_{12\_im} = \sum_{n=1}^{4} a2_i d2_i$$

$$\tilde{r}_{21\_re} = \sum_{n=1}^{4} b_i c_i, \quad \tilde{r}_{21\_im} = \sum_{n=1}^{4} c_i d_i$$

$$\tilde{r}_{22\_re} = \sum_{n=1}^{4} b2_i c2_i, \quad \tilde{r}_{22\_im} = \sum_{n=1}^{4} c2_i d2_i$$

Le code VHDL peut être trouvé en annexe B 2.3.

Opérande	1[er]cycle	2[e]cycle	3[e]cycle	4[e]cycle
a	$Re(h_{11})$	$Im(h_{11})$	$Re(h_{12})$	$Im(h_{12})$
b	$Re(x_{11})$	$Im(x_{11})$	$Re(x_{21})$	$Im(x_{21})$
c	$Re(h_{21})$	$Im(h_{21})$	$Re(h_{22})$	$Im(h_{22})$
d	$Im(x_{11})$	$Re(x_{11})$	$Im(x_{21})$	$Re(x_{21})$
a2	$Re(h_{11})$	$Im(h_{11})$	$Re(h_{12})$	$Im(h_{12})$
b2	$Re(x_{12})$	$Im(x_{12})$	$Re(x_{22})$	$Im(x_{22})$
c2	$Re(h_{21})$	$Im(h_{21})$	$Re(h_{22})$	$Im(h_{22})$
d2	$Im(x_{12})$	$Re(x_{12})$	$Im(x_{22})$	$Re(x_{22})$

TABLE 3.2 – Tableau d'affectation des opérandes au canal

3.4 Résultats des simulations :

Dans le cas de la chaine à modulateur QAM, nous avons injecté les signaux en entrée [input1,input2] ([00 , 01] ;[11, 10]) qui correspondent aux séquences [0100] et [1011] respectivement. Pour chaque signal le résultat de décodage aparait aprés 17 cycles d'horloges (voir la figure 3.3).

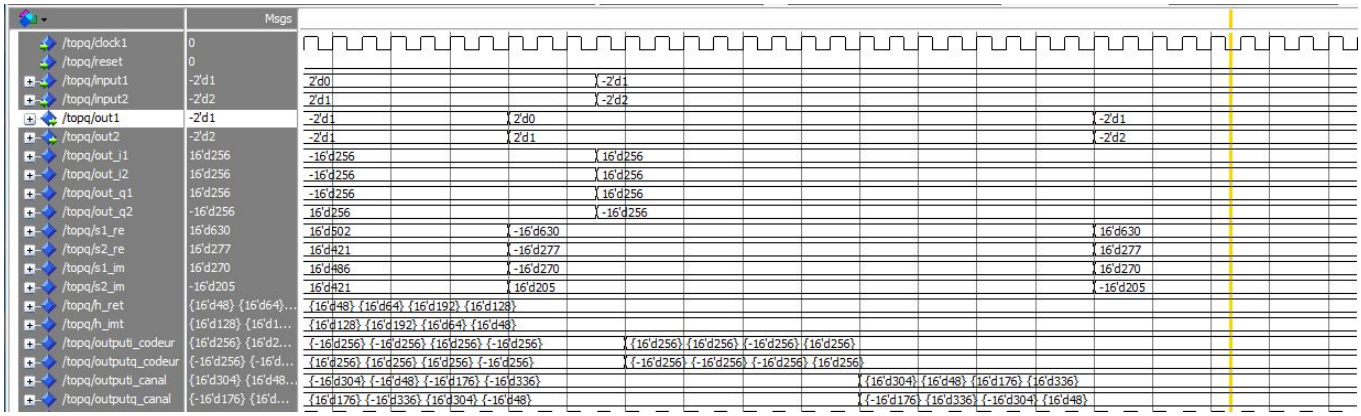

FIGURE 3.3 – Resultats de la simulation de QAM

Pour une modulation BPSK, Les signaux injectés en entrée sont [input1,input2] ([0,1] ;[1,1]) qui correspondent aux séquences [10] et [11] respectivement (voir la figure 3.4).

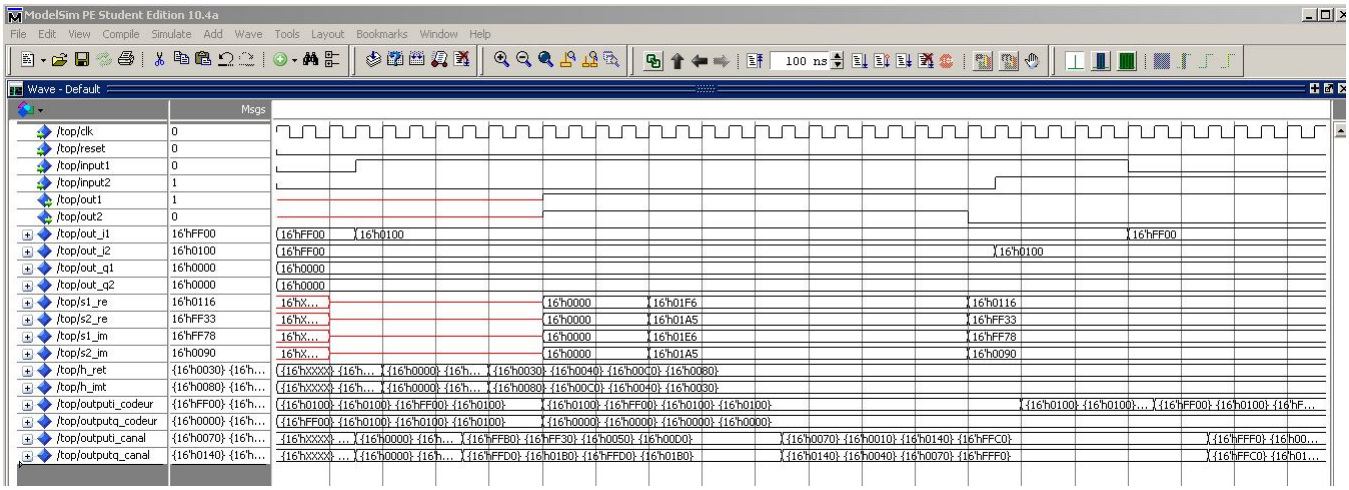

FIGURE 3.4 – Résultats de la simulation de BPSK

3.5 Implémentation sur FPGA

3.5.1 Premier contact :

PGA ML501 est un diapositive semi-conducteur programmable et reprogrammable. C'est le simulateur hard à prix d'implémentation et temps de cycle bas.
Outils nécessaires pour l'implémentation :

1. Xilinx ISE web pack 13.2 pour design, synthesis et implementation.
2. Carte FPGA ML501

3.5.2 Caractéristiques :

Les tableaux ci dessous résument les différentes horloges de l'FPGA ML501 et quelques pins ainsi que quelques LED dont on a utilisé.

Label	Clock Name	FPGA Pin	Description
X1	USER_CLK	AD8	100 MHz single-ended
U8	CLK_33MHZ_FPGA	AB12	33 MHz single-ended
U8	CLK_27MHZ_FPGA	AD13	27 MHz single-ended
U8	CLK_DIFF_FPGA_P	E16	200 MHz differential pair (pos)
U8	CLK_DIFF_FPGA_N	E17	200 MHz differential pair (neg)

FIGURE 3.5 – Horloges du ML501

Reference Designator	Label/Definition	Color	FPGA Pin	Buffered
DS20	LED North	Green	Y8	Yes
DS21	LED East	Green	Y18	Yes
DS22	LED South	Green	AA8	Yes
DS23	LED West	Green	AA18	Yes
DS24	LED Center	Green	T22	Yes
DS17	GPIO LED 0	Green	E13	Yes
DS16	GPIO LED 1	Green	D14	Yes
DS15	GPIO LED 2	Green	E12	Yes
DS14	GPIO LED 3	Green	F12	Yes
DS13	GPIO LED 4	Green	D15	No
DS12	GPIO LED 5	Green	E15	No
DS11	GPIO LED 6	Green	E10	No
DS10	GPIO LED 7	Green	E11	No
D56	Error 1	Red	N4	No
D55	Error 2	Red	P5	No

FIGURE 3.6 – Leds principales du ML501

SW4	FPGA Pin
GPIO_DIP_SW1	U4
GPIO_DIP_SW2	V3
GPIO_DIP_SW3	T4
GPIO_DIP_SW4	T5
GPIO_DIP_SW5	U6
GPIO_DIP_SW6	U5
GPIO_DIP_SW7	U7
GPIO_DIP_SW8	T7

FIGURE 3.7 – Switch utilisé du ML501

3.5.3 Résultats de l'implémentation :

Les Schémas RTL propre à la chaine de transmission pour les deux modulations (QAM et BPSK) sont illustrés sur les figures (3.8) et (3.9), la figure (3.10) représente le shéma du décodeur.

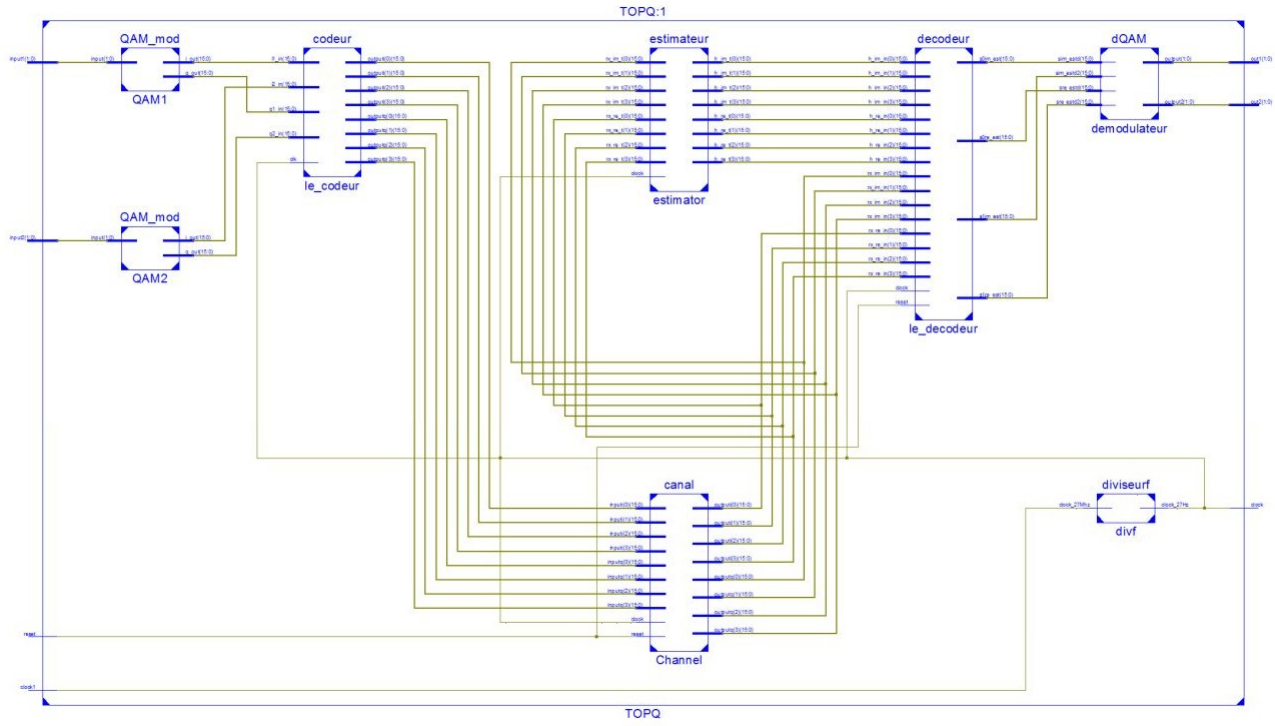

FIGURE 3.8 – Schéma RTL de la chaine de transmission à modulateur QAM

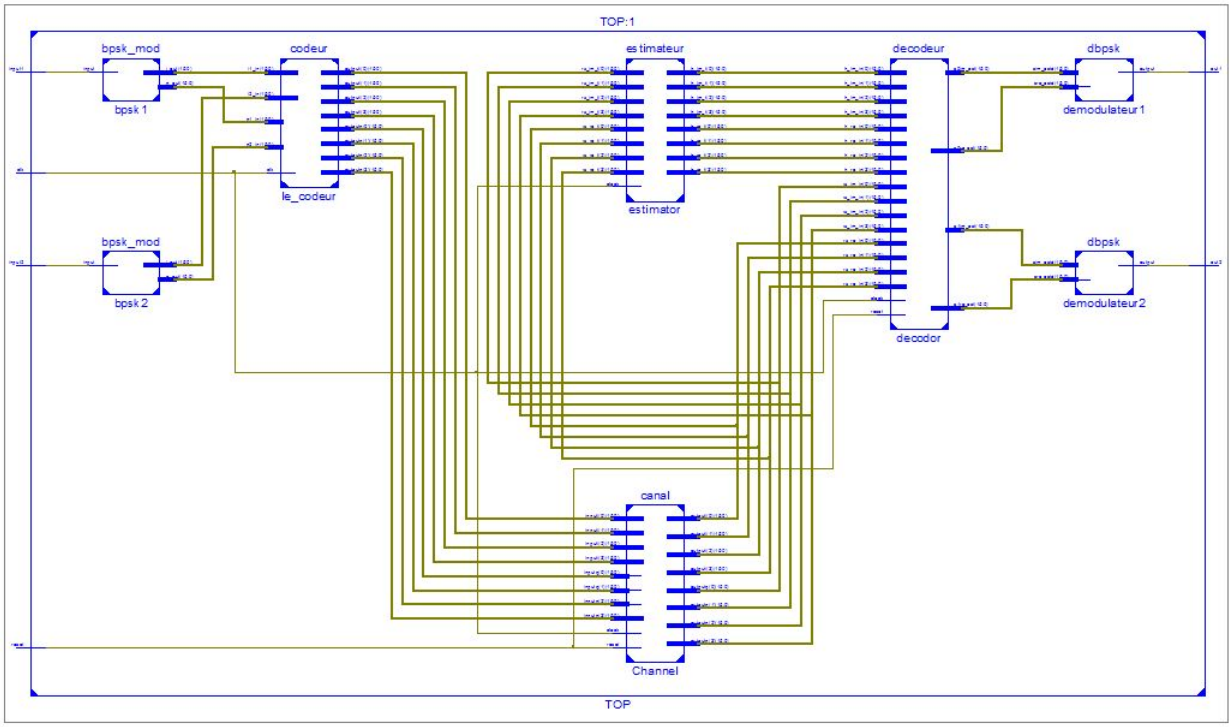

FIGURE 3.9 – Schéma RTL de la chaine de transmission à modulateur BPSK

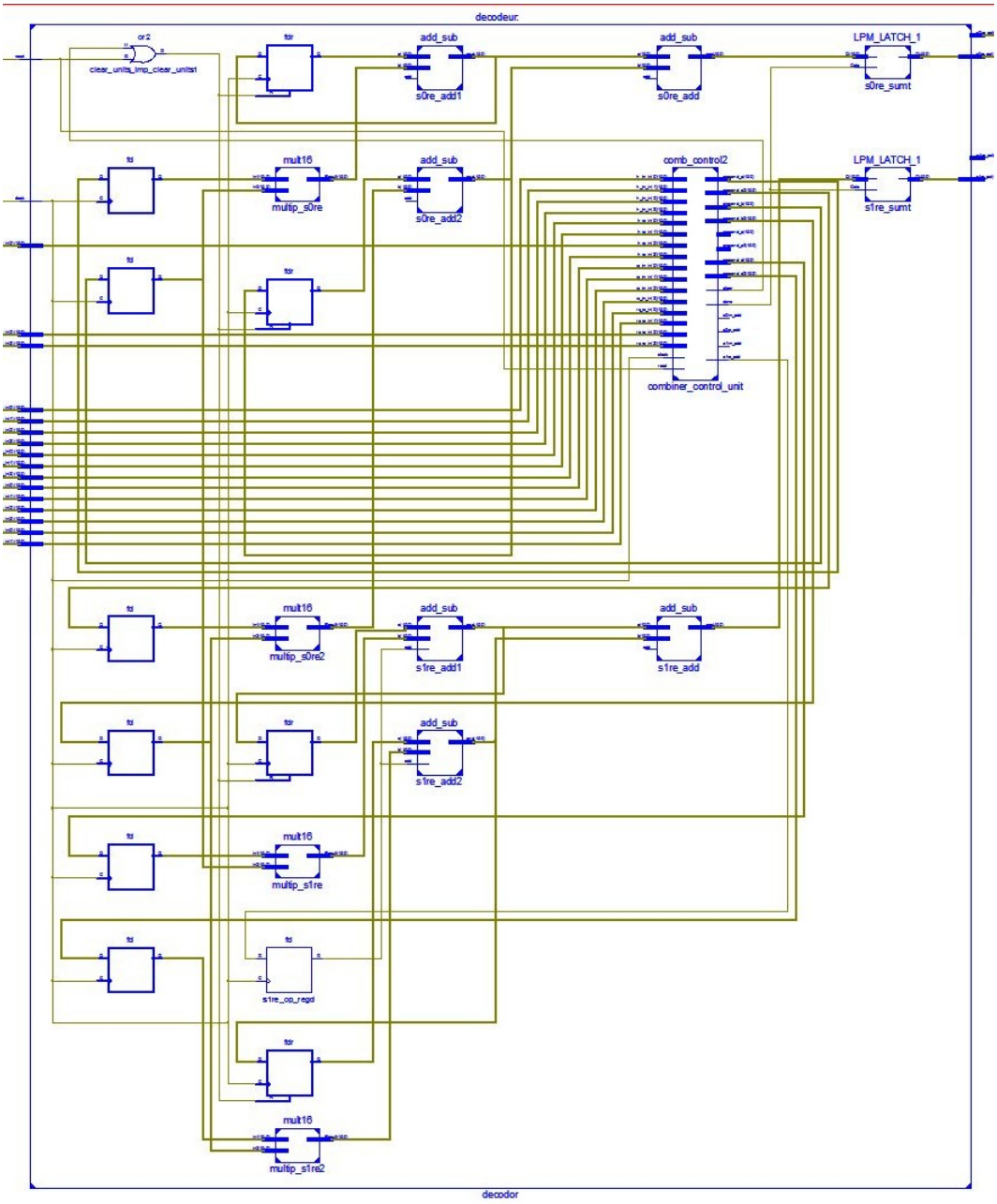

FIGURE 3.10 – Schéma RTL du Décodeur

La figure 3.11 représente le rapport du design fourni par l'outil XILINX, on remarque que les ressources utilisés sont raisonnables.

TOPQ Project Status (06/14/2017 - 17:06:37)			
Project File:	QAM.xise	**Parser Errors:**	No Errors
Module Name:	TOPQ	**Implementation State:**	Synthesized
Target Device:	xc5vlx50-1ff676	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	288 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	914	28800	3%
Number of Slice LUTs	1123	28800	3%
Number of fully used LUT-FF pairs	626	1411	44%
Number of bonded IOBs	10	440	2%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	40	48	83%

FIGURE 3.11 – Rapport de Synthèse

Nous avons choisi pour notre test :

- Horloge : 27MHz, à l'aide d'un diviseur de fréquence on a réduit la fréquence à 27 Hz afin de vérifier le bon fonctionnement du code.
- Switch :
  1. U4 : Input1 (1).
  2. V3 : Input1 (0).
  3. T4 : Input2 (1).
  4. T5 : Input2 (0).
  5. T7 : pour le reset.
- Leds :
  1. D15 : output1 (1).
  2. E15 : output1 (0).
  3. E10 : output2 (1).
  4. E11 : output2 (0).

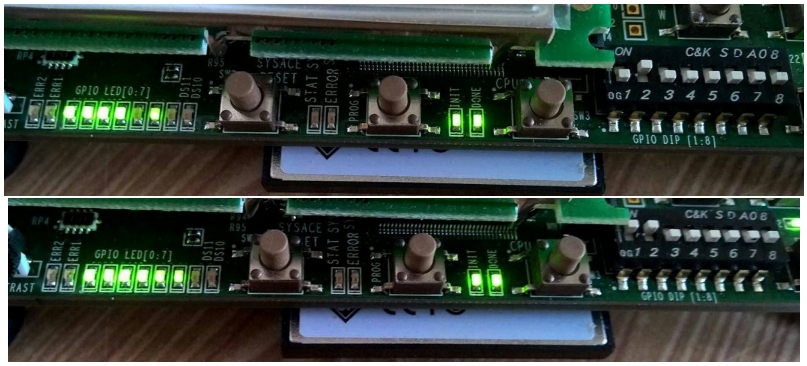

FIGURE 3.12 – Photographie du Kit FPGA durant l'implémentation

3.6 Conclusion

Dans ce chapitre nous avons commencé par une simulation à l'aide de l'outil Modelsim. Après validation des résultats, l'outil Xilinx ISE nous a permis de faire la synthèse de la chaine de transmission, la rapport de synthèse a montré que les ressources utilisées dans la chaine de transmission sont raisonnables comparé aux ressources disponibles sur la carte FPGA ce qui permet une amélioration en augmentant le nombre d'antennes, rendre le canal plus complexe ...etc

Conclusion et perspectives

Nous avons exposé les principes de base des systèmes de communication MiMo, nous avons étudié les codes spatio-temporels en bloc, un accent particulier a été mis sur la méthode d'Alamouti.

Nous avons développé un support logiciel simulant une chaine de communications MIMO sous Matlab/Simulink qui nous a permis de tester et valider le codeur/décodeur d'Alamouti sous différents schémas de modulation.

Nous avons également développé en Vhdl les codes modélisant les circuits numériques du codeur/décodeur d'Alamouti ainsi que de quelques circuits associés tels que le simulateur de canal, l'estimateur de canal .

L'étude comportementale de notre design à l'aide de modelsim a montré l'efficacité de la technique d'Alamouti et la justesse de l'architecture conçue à cet effet.

Nous avons conclu notre travail par la synthèse et la projection de notre design sur circuit configurable, en l'occurrence le Fpga Virtex5, à l'aide de l'outil Ise de Xilinx.

En perspective nous suggérons que la méthodologie utilisée pour notre design soit étendue aux cas de l'utilisation d'un nombre d'antennes supérieur à 2 aussi bien en émission qu'en réception et de leurs associer des schémas de modulation à plusieurs points.
Il est possible également de recourir aux techniques itératives qui connaissent actuellement un engouement majeur pour décoder les codes spatio-temporels.

Bibliographie

[1] M. JANKIRAMAN «*space-time codes and MIMO systems*» , Artech House Universal Personal Communications series,edition 2004.

[2] M.Farhad «*MATLAB - A Fundamental Tool for Scientific Computing and Engineering Applications*» - *Volume 2*,book edited by Vasilios N. Katsikis, ISBN $978-953-51-0751-4$, Published : September 26, 2012 under CC BY 3.0 license.

[3] Modelsim, Mentor, a siemens business :
https ://www.mentor.com/products/fv/modelsim/.

[4] F.Guillaume «*Codage spatio-temporel et techniques de décodage itératives pour systèmes multi-antennes*»,Thèse de doctorat,Université de Limoges, Le 4 Juillet 2006.

[5] R. Ouertani, «*Algorithmes de décodage pour les systèmes multi-antennes à complexité réduite*»,Théorie de l'information [cs.IT]. Télécom ParisTech, 2009. Français. <pastel-00718214>

[6] K.U. Keshari,«*A Novel Rate-2 Space Time Block Code & implementation of its decoder*», Thèse de doctorat, Thapar University Patiala-147004 (PUNJAB), Juillet 2014.

[7] L. SCREMIN, «*Codage spatio-T pour Les systèmes Multi-Antennes de communications Sans-Fil*»,Thèse de doctorat, école polytechnique de Montréal, Juin 2000.

[8] *Chapitre 1. Modulations multiporteuses et non linéarités,*
http ://www.becoz.org/these/memoirehtml/ch05s02.html

[9] M. SAYED HASSAN, «*Codage spatio-temporel optimisé pour une concaténation série avec les codes correcteurs d'erreurs*»,Thèse de doctorat, Télécom Bretagne, $22/10/2010$

Code Matlab

Décodeur :

```
function [out]= decodeur(in,nb,x_t)
in_i=1;
out_i=1;

for block_i=1:(nb+ length(x_t)):columns(in)
  a=out_i+1;
  in_i1=in_i+1;
  y_t= in(:,[in_i1 in_i1 in_i1 in_i1]);

%iteration
  for sym_i=1:2:nb
   y=in(:,[in_i1 in_i1]);
    s0_squig=conj(H_est(1,1))*y(1,1)+H_est(1,2)*conj(y(1,2))+conj(H_est(2,1))*y(2,1)+H_est(2,2)*conj(y(2,2));

    s1_squig=conj(H_est(1,2))*y(1,1)-H_est(1,1)*conj(y(1,2))+conj(H_est(2,2))*y(2,1)-H_est(2,1)*conj(y(2,2));
  if real(s0_squig)<0
      out(a)=0;
    else
      out(a)=1;
    end
   if real(s1_squig)<0
       out(out_i++)=0;
   else
       out(out_i++)=1;
   end
   end %fin decodage
end %Fin de loop
end
```

Estimateur :

```
1    function h_est = chan_est(y_t,x_t)
2
3 -  h_est=y_t* x_t'*inv(x_t*x_t');
4
5    end
```

Code VHDL

Librairie mes_types :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
use ieee.std_logic_SIGNED.all;


package mes_types is

type matrix is array(3 downto 0) of std_logic_vector(15 downto 0);
end mes_types;

package body mes_types is


end mes_types;
```

Modulateur :

Modulateur BPSK :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_SIGNED.all;

entity bpsk_mod is
port(input :in std_logic;
  i_out, q_out : out std_logic_vector(15 downto 0)
  );
end bpsk_mod;

architecture Behavioral of bpsk_mod is

begin
with input select
  -----  1+i0   § -1+i0
  i_out <="0000000100000000" when '1',    --------- 1
          "1111111100000000" when others;  -------- -1
  q_out <="0000000000000000";


end Behavioral;
```

Modulateur QAM :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

library work;
use work.mes_types.all;

entity QAM_mod is port(
  input        : in std_logic_vector(1 downto 0);
  i_out,q_out  : out std_logic_vector(15 downto 0)
  );
  end QAM_mod;

architecture Behavioral of QAM_mod is
  begin
my_process: process(input)
begin
  case (input) is
    when"11" =>
       i_out <="0000000100000000";
       q_out <="0000000100000000";
    when "10" =>
       i_out <="0000000100000000";
       q_out <="1111111100000000";
    when "01" =>
       i_out  <="1111111100000000";
       q_out  <="0000000100000000";
    when "00" =>
       i_out  <="1111111100000000";
       q_out  <="1111111100000000";
    when others =>
       i_out  <="0000000000000000";
       q_out  <="0000000000000000";
  end case;
end process;
  end Behavioral;
```

Codeur :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;
use ieee.std_logic_SIGNED.all;

library work;
use work.mes_types.all;
entity codeur is
port (
            clk         : in std_logic;
        i1_in,i2_in : in std_logic_vector(15 downto 0);
        q1_in,q2_in : in std_logic_vector(15 downto 0);
            outputi: out matrix;
            outputq: out matrix
        );
end codeur;

architecture Behavioral of codeur is
signal compt : integer:=0;
signal C : matrix:=("0000000100000000","0000000100000000","1111111100000000","0000000100000000");
signal C2: matrix:=("1111111100000000","0000000100000000","0000000100000000","0000000100000000");
begin

process(clk)
begin
if (Clk'event and Clk='1') then
if (Compt<10) then
    Compt <=Compt+1;
outputi<=C;
outputq<=C2;
else
outputi(0)<=i1_in;
outputi(1)<=-i2_in;
outputi(2)<=i2_in;
outputi(3)<=i1_in;
outputq(0)<=q1_in;
outputq(1)<=q2_in;
outputq(2)<=q2_in;
outputq(3)<=-q1_in;
    end if;
    end if;
  end process;
end Behavioral;
```

Simulateur de Canal :

Canal :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

library work;
use work.mes_types.all;

entity canal is
port(
reset  : in std_logic;
clock  : in std_logic;
inputi : in matrix;
inputq : in matrix;
outputi: out matrix;
outputq: out matrix

);
end canal;

architecture Behavioral of canal is
component canal_control2
port(
     clock : in std_logic;
      reset:  in std_logic;
      rx_re_in: in matrix;
      rx_im_in: in matrix;
      --
      operand_a  :out std_logic_vector(15 downto 0);
      operand_b  :out std_logic_vector(15 downto 0);
      operand_c  :out std_logic_vector(15 downto 0);
      operand_d  :out std_logic_vector(15 downto 0);
      operand_b2 :out std_logic_vector(15 downto 0);
      operand_d2 :out std_logic_vector(15 downto 0);
      ---
```

```
      ---
      r0re_add,r0im_add: out std_logic;
      r1re_add,r1im_add: out std_logic;
     r2re_add,r2im_add: out std_logic;
      r3re_add,r3im_add: out std_logic;

      done : out std_logic;
      clear: out std_logic
      );

end component;

component add_sub
port(
      a: in std_logic_vector(15 downto 0);
       b: in std_logic_vector(15 downto 0);
       add:in std_logic;
       ans:out std_logic_vector(15 downto 0)
      );
end component;
component mult16
port
   (
      in1, in2: in std_logic_vector(15 downto 0);
      Result: out std_logic_vector(15 downto 0)
   );
end component;
--------------------------------------------------------------------------------
-------------------------------------------------DECLARATION DES SIGNAUX-----------
--------------------------------------------------------------------------------

```

```
100       signal r0re_sum: std_logic_vector(15 downto 0);
101       signal r0im_sum: std_logic_vector(15 downto 0);
102       signal r1re_sum: std_logic_vector(15 downto 0);
103       signal r1im_sum: std_logic_vector(15 downto 0);
104      signal r2re_sum: std_logic_vector(15 downto 0);
105       signal r2im_sum: std_logic_vector(15 downto 0);
106       signal r3re_sum: std_logic_vector(15 downto 0);
107       signal r3im_sum: std_logic_vector(15 downto 0);
108       ---
109       signal r0re_total: std_logic_vector(15 downto 0);
110       signal r0im_total: std_logic_vector(15 downto 0);
111       signal r1re_total: std_logic_vector(15 downto 0);
112       signal r1im_total: std_logic_vector(15 downto 0);
113      signal r2re_total: std_logic_vector(15 downto 0);
114       signal r2im_total: std_logic_vector(15 downto 0);
115       signal r3re_total: std_logic_vector(15 downto 0);
116       signal r3im_total: std_logic_vector(15 downto 0);
117       ----
118       signal r0re_op_regd :std_logic;
119       signal r0im_op_regd :std_logic;
120       signal r1re_op_regd :std_logic;
121       signal r1im_op_regd :std_logic;
122      signal r2re_op_regd :std_logic;
123       signal r2im_op_regd :std_logic;
124       signal r3re_op_regd :std_logic;
125       signal r3im_op_regd :std_logic;
126
127          signal done_i: std_logic;
128  ----------------------------------------------------
129  begin
130
131  clear_units <= reset or clear_control;
132
133  canal_control_unit: canal_control2 port map(
134
```

```
135      clock => clock,
136       reset => reset,
137       rx_re_in=>inputi,
138       rx_im_in=>inputq,
139       --
140       operand_a  => op_at,
141       operand_b  => op_bt,
142       operand_c  => op_ct,
143       operand_d  => op_dt,
144       operand_b2 => op_b2t,
145       operand_d2 => op_d2t,
146       ---
147       r0re_add=>r0re_op,
148       r0im_add=>r0im_op,
149       r1re_add=>r1re_op,
150       r1im_add=>r1im_op,
151      r2re_add=>r2re_op,
152       r2im_add=>r2im_op,
153       r3re_add=>r3re_op,
154       r3im_add=>r3im_op,
155       done=>done_i,
156       clear=>clear_control
157       );
158
```

```
process (clock)
begin
if (clock'event and clock ='1') then
 r0re_op_regd <= r0re_op;
 r0im_op_regd <= r0im_op;
 r1re_op_regd <= r1re_op;
 r1im_op_regd <= r1im_op;
 r2re_op_regd <= r2re_op;
 r2im_op_regd <= r2im_op;
 r3re_op_regd <= r3re_op;
 r3im_op_regd <= r3im_op;
 op_a <=op_at;
 op_b <=op_bt;
 op_c <=op_ct;
 op_d <=op_dt;
 op_b2 <=op_b2t;
 op_d2 <=op_d2t;
end if;
end process;
--------
----registre et multiplicateurs
--------
multip_r0re: mult16 port map
   (
      in1=> op_a,
      in2=> op_b ,
      Result=>r0re_prod
   );
 multip_r1re: mult16 port map
   (
      in1=> op_a,
      in2=> op_b2 ,
      Result=>r1re_prod
   );
---------------------------------
```

```
194   multip_r0im: mult16 port map
195     (
196       in1=> op_a,
197       in2=> op_d ,
198       Result=>r0im_prod
199     );
200   multip_r1im: mult16 port map
201     (
202       in1=> op_a,
203       in2=> op_d2,
204       Result=>r1im_prod
205     );
206   ----------------------------------
207   multip_r2re: mult16 port map
208     (
209       in1=> op_b,
210       in2=> op_c ,
211       Result=>r2re_prod
212     );
213   multip_r3re: mult16 port map
214     (
215       in1=> op_c,
216       in2=> op_b2 ,
217       Result=>r3re_prod
218     );
219   --------------------------------
220   multip_r2im: mult16 port map
221     (
222       in1=> op_c,
223       in2=> op_d ,
224       Result=>r2im_prod
225     );
```

```
r2re_add: add_sub port map(
   a => r2re_total,
    b => r2re_prod,
    add =>r2re_op_regd,
    ans => r2re_sum
    );
 r2im_add: add_sub port map(
   a => r2im_total,
    b => r2im_prod,
    add =>r2im_op_regd,
    ans => r2im_sum
    );

 r3re_add: add_sub port map(
   a => r3re_total,
    b => r3re_prod,
    add =>r3re_op_regd,
    ans => r3re_sum
    );
 r3im_add: add_sub port map(
   a => r3im_total,
    b => r3im_prod,
    add =>r3im_op_regd,
    ans => r3im_sum
    );
--------------------------------------------------------------------------------
-----------------------registre-----------------------------------------------
--------------------------------------------------------------------------------
```

```
r0re_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r0re_total <=  x"0000";
      else
      r0re_total <= r0re_sum;
      end if;
      end if;
      end process;

r0im_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r0im_total <=  x"0000";
      else
      r0im_total <= r0im_sum;
      end if;
      end if;
      end process;
 --
r1re_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r1re_total <=  x"0000";
      else
      r1re_total <= r1re_sum;
      end if;
      end if;
      end process;

```

```
r1im_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r1im_total <=  x"0000";
      else
      r1im_total <= r1im_sum;
      end if;
      end if;
      end process;
r2re_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r2re_total <=  x"0000";
      else
      r2re_total <= r2re_sum;
      end if;
      end if;
      end process;

r2im_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r2im_total <=  x"0000";
      else
      r2im_total <= r2im_sum;
      end if;
      end if;
      end process;

```

```
r3re_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r3re_total <=  x"0000";
      else
      r3re_total <= r3re_sum;
      end if;
      end if;
      end process;

r3im_reg:process (clock)
    begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      r3im_total <=  x"0000";
      else
      r3im_total <= r3im_sum;
      end if;
      end if;
      end process;
--------------------------------------------------------------------------------
--------------------output assignements-----------------------------------------
--------------------------------------------------------------------------------
process(clock)
begin

if (clock'event and clock ='1') then
if (done_i='1') then
outputi(0)<=r0re_sum;
outputi(1)<=r1re_sum;
outputi(2)<=r2re_sum;
outputi(3)<=r3re_sum;

outputq(0)<=r0im_sum;
outputq(1)<=r1im_sum;
outputq(2)<=r2im_sum;
outputq(3)<=r3im_sum;
end if;
end if;
end process;
end Behavioral;
```

Contrôle du Canal :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_SIGNED.all;

library work;
use work.mes_types.all;

entity canal_control2 is
port(
     clock   : in std_logic;
          reset   : in std_logic;
          rx_re_in: in matrix;
          rx_im_in: in matrix;
          --
          operand_a  :out std_logic_vector(15 downto 0);
          operand_b  :out std_logic_vector(15 downto 0);
          operand_c  :out std_logic_vector(15 downto 0);
          operand_d  :out std_logic_vector(15 downto 0);
          operand_b2 :out std_logic_vector(15 downto 0);
          operand_d2 :out std_logic_vector(15 downto 0);
          ---
          r0re_add,r0im_add: out std_logic;
          r1re_add,r1im_add: out std_logic;
     r2re_add,r2im_add: out std_logic;
          r3re_add,r3im_add: out std_logic;

          done : out std_logic;
          clear: out std_logic
          );

end canal_control2;

architecture Behavioral of canal_control2 is

type state_type is (st_rst,st1,st2,st3,st4);
     signal state, next_state : state_type;
     signal op_a_i :std_logic_vector(15 downto 0);
     signal op_b_i :std_logic_vector(15 downto 0);
     signal op_c_i :std_logic_vector(15 downto 0);
     signal op_d_i :std_logic_vector(15 downto 0);
     signal op_b2_i :std_logic_vector(15 downto 0);
     signal op_d2_i :std_logic_vector(15 downto 0);
```

```
    signal r0re_add_i,r0im_add_i:std_logic;
    signal r1re_add_i,r1im_add_i: std_logic;
    signal r2re_add_i,r2im_add_i:std_logic;
    signal r3re_add_i,r3im_add_i: std_logic;

    signal rx_re_reg:  matrix;
    signal rx_im_reg:  matrix;

    signal h_re_reg : matrix := ("0000000000110000","0000000001000000","0000000011000000","0000000010000000");
    signal h_im_reg : matrix := ("0000000010000000","0000000011000000","0000000001000000","0000000000110000");
         -- (0.2+i0.5) (0.3+i0.8) (0.8+i0.3) (0.5+i0.2)

    signal done_i  :std_logic;
    signal clear_i :std_logic;
begin
  input_regs: process (clock,reset)
   begin
     if (clock'event and clock ='1') then  --- effacement de data
        if (reset='1') then
              rx_re_reg <=(others =>x"0000");
              rx_im_reg <=(others =>x"0000");

                else if (state = st4) or (state= st_rst) then
                 rx_re_reg <=rx_re_in;
                 rx_im_reg <=rx_im_in;

            end if;
         end if;
         end if;
        end process;
  sync_proc: process (clock,reset)
   begin
     if (clock'event and clock ='1') then
        if (reset='1') then
         state  <= st_rst;
         r0re_add <= '1';
         r0im_add <= '1';
         r1re_add <= '1';
         r1im_add <= '1';
         r2re_add <= '1';
         r2im_add <= '1';
         r3re_add <= '1';
         r3im_add <= '1';
```

```
              operand_a  <= x"0000";
              operand_b  <= x"0000";
              operand_c  <= x"0000";
              operand_d  <= x"0000";
              operand_b2 <= x"0000";
              operand_d2 <= x"0000";

         done <= '0';
         clear<= '1';
 else
    state     <= next_state;
    r0re_add  <= r0re_add_i;
    r0im_add  <= r0im_add_i;
    r1re_add  <= r1re_add_i ;
    r1im_add  <= r1im_add_i;
    r2re_add  <= r2re_add_i;
    r2im_add  <= r2im_add_i;
    r3re_add  <= r3re_add_i ;
    r3im_add  <= r3im_add_i;
    done      <= done_i;
    clear     <= clear_i;
    operand_a <= op_a_i;
    operand_b <=op_b_i;
    operand_c <=op_c_i;
    operand_d <=op_d_i;
    operand_b2 <=op_b2_i;
    operand_d2 <=op_d2_i;

          end if;
          end if;
end process;

output_canal: process (state,h_re_reg,h_im_reg,rx_re_reg,rx_im_reg)
begin
case(state)is
```

```
 when st_rst =>
  op_a_i      <=x"0000";
  op_b_i      <=x"0000";
  op_c_i      <=x"0000";
  op_d_i      <=x"0000";
  op_b2_i     <=x"0000";
  op_d2_i     <=x"0000";
  r0re_add_i <= '1';
  r0im_add_i <= '1';
  r1re_add_i <= '1';
  r1im_add_i <= '1';
  r2re_add_i <= '1';
  r2im_add_i <= '1';
  r3re_add_i <= '1';
  r3im_add_i <= '1';
  done_i      <='0';
  clear_i     <='1';

---- 1 er état de la machine d'état
when st1 =>
  op_a_i      <=h_re_reg(0);
  op_b_i      <=rx_re_reg(0);
  op_c_i      <=h_re_reg(2);
  op_d_i      <=rx_im_reg(0);
  op_b2_i     <=rx_re_reg(1);
  op_d2_i     <=rx_im_reg(1);
  r0re_add_i <= '1';
  r0im_add_i <= '1';
  r2re_add_i <= '1';
  r2im_add_i <= '1';
  r1re_add_i <= '1';
  r1im_add_i <= '1';
  r3re_add_i <= '1';
  r3im_add_i <= '1';
```

```
  done_i     <= '1';
  clear_i    <= '1';
---- 2 eme état de la machine d'état
when st2 =>
  op_a_i     <=h_im_reg(0);
  op_b_i     <=rx_im_reg(0);
  op_c_i     <=h_im_reg(2);
  op_d_i     <=rx_re_reg(0);
  op_b2_i    <=rx_im_reg(1);
  op_d2_i    <=rx_re_reg(1);
  r0re_add_i <= '0';
  r0im_add_i <= '1';
  r2re_add_i <= '0';
  r2im_add_i <= '1';
  r1re_add_i <= '0';
  r1im_add_i <= '1';
  r3re_add_i <= '0';
  r3im_add_i <= '1';

  done_i     <= '0';
  clear_i    <= '0';
---- 3 eme état de la machine d'état
when st3 =>
  op_a_i     <=h_re_reg(1);
  op_b_i     <=rx_re_reg(2);
  op_c_i     <=h_re_reg(3);
  op_d_i     <=rx_im_reg(2);
  op_b2_i    <=rx_re_reg(3);
  op_d2_i    <=rx_im_reg(3);
  r0re_add_i <= '1';
  r0im_add_i <= '1';
  r2re_add_i <= '1';
  r2im_add_i <= '1';
  r1re_add_i <= '1';
  r1im_add_i <= '1';
  r3re_add_i <= '1';
  r3im_add_i <= '1';
  done_i     <='0';
  clear_i    <='0';
---- 4 eme état de la machine d'état
```

```
when st4 =>
  op_a_i      <=h_im_reg(1);
  op_b_i      <=rx_im_reg(2);
  op_c_i      <=h_im_reg(3);
  op_d_i      <=rx_re_reg(2);
  op_b2_i     <=rx_im_reg(3);
  op_d2_i     <=rx_re_reg(3);
  r0re_add_i <= '0';
  r0im_add_i <= '1';
  r2re_add_i <= '0';
  r2im_add_i <= '1';
  r1re_add_i <= '0';
  r1im_add_i <= '1';
  r3re_add_i <= '0';
  r3im_add_i <= '1';

  done_i      <='0';
  clear_i     <='0';

                end case;

 end process;
 process(state)
  begin
  next_state <= state;
  case(state)is
   when st_rst =>next_state<=st1;
        when st1=>next_state<=st2;
        when st2=>next_state<=st3;
        when st3=>next_state<=st4;
        when st4=>next_state<=st1;
        when others =>
        next_state <=st_rst;
        end case;
end process;
end Behavioral;
```

Décodeur :

Décodeur :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.ALL;
4  use ieee.std_logic_signed.all;
5  library work;
6  use work.mes_types.all;
7
8  entity decodeur is
9  port(
10       clock : in std_logic;
11        reset : in std_logic;
12        rx_re_in:in matrix;
13        rx_im_in:in matrix;
14        h_re_in: in matrix;
15        h_im_in: in matrix;
16
17        s0re_est:out std_logic_vector(15 downto 0);
18        s0im_est:out std_logic_vector(15 downto 0);
19        s1re_est:out std_logic_vector(15 downto 0);
20        s1im_est:out std_logic_vector(15 downto 0)
21
22        );
23 end decodeur;
24
25 architecture Behavioral of decodeur is
26
27       signal s0re_op :std_logic;
28        signal s0im_op :std_logic;
29        signal s1re_op :std_logic;
30        signal s1im_op :std_logic;
31 --
```

```
 signal clear_control :std_logic;
 signal clear_units   :std_logic;
signal op_a :std_logic_vector(15 downto 0);
 signal op_b :std_logic_vector(15 downto 0);
 signal op_c :std_logic_vector(15 downto 0);
 signal op_d :std_logic_vector(15 downto 0);
 signal op_a2 :std_logic_vector(15 downto 0);
 signal op_b2 :std_logic_vector(15 downto 0);
 signal op_c2 :std_logic_vector(15 downto 0);
 signal op_d2 :std_logic_vector(15 downto 0);
signal op_at :std_logic_vector(15 downto 0);
 signal op_bt :std_logic_vector(15 downto 0);
 signal op_ct :std_logic_vector(15 downto 0);
 signal op_dt :std_logic_vector(15 downto 0);
 signal op_a2t :std_logic_vector(15 downto 0);
 signal op_b2t :std_logic_vector(15 downto 0);
 signal op_c2t :std_logic_vector(15 downto 0);
 signal op_d2t :std_logic_vector(15 downto 0);
 ---
 signal s0re_prod: std_logic_vector(15 downto 0);
 signal s0im_prod:  std_logic_vector(15 downto 0);
 signal s1re_prod:  std_logic_vector(15 downto 0);
 signal s1im_prod:  std_logic_vector(15 downto 0);
 signal s0re_prod2: std_logic_vector(15 downto 0);
 signal s0im_prod2:  std_logic_vector(15 downto 0);
 signal s1re_prod2:  std_logic_vector(15 downto 0);
 signal s1im_prod2:  std_logic_vector(15 downto 0);
 ---
 signal s0re_sum1: std_logic_vector(15 downto 0);
 signal s0im_sum1: std_logic_vector(15 downto 0);
 signal s1re_sum1: std_logic_vector(15 downto 0);
 signal s1im_sum1: std_logic_vector(15 downto 0);
signal s0re_sum2: std_logic_vector(15 downto 0);
 signal s0im_sum2: std_logic_vector(15 downto 0);
 signal s1re_sum2: std_logic_vector(15 downto 0);
 signal s1im_sum2: std_logic_vector(15 downto 0);
 ----
```

```
     signal s0re_sum : std_logic_vector(15 downto 0);
      signal s0im_sum : std_logic_vector(15 downto 0);
      signal s1re_sum : std_logic_vector(15 downto 0);
      signal s1im_sum : std_logic_vector(15 downto 0);
      signal s0re_sumt: std_logic_vector(15 downto 0);
      signal s0im_sumt: std_logic_vector(15 downto 0);
      signal s1re_sumt: std_logic_vector(15 downto 0);
      signal s1im_sumt: std_logic_vector(15 downto 0);
      ---
      signal s0re_total : std_logic_vector(15 downto 0);
      signal s0im_total : std_logic_vector(15 downto 0);
      signal s1re_total : std_logic_vector(15 downto 0);
      signal s1im_total : std_logic_vector(15 downto 0);
     signal s0re_total2: std_logic_vector(15 downto 0);
      signal s0im_total2: std_logic_vector(15 downto 0);
      signal s1re_total2: std_logic_vector(15 downto 0);
      signal s1im_total2: std_logic_vector(15 downto 0);

      signal s0re_op_regd :std_logic;
      signal s0im_op_regd :std_logic;
      signal s1re_op_regd :std_logic;
      signal s1im_op_regd :std_logic;
     signal done_i: std_logic;
------
----declaration des composants
-----
component comb_control2
port(
     clock : in std_logic;
      reset : in std_logic;
      rx_re_in: in matrix;
      rx_im_in: in matrix;
      h_re_in :  in matrix;
      h_im_in :  in  matrix;
      --
```

```
104         operand_a  :out std_logic_vector(15 downto 0);
105         operand_b  :out std_logic_vector(15 downto 0);
106         operand_c  :out std_logic_vector(15 downto 0);
107         operand_d  :out std_logic_vector(15 downto 0);
108        operand_a2 :out std_logic_vector(15 downto 0);
109         operand_b2 :out std_logic_vector(15 downto 0);
110         operand_c2 :out std_logic_vector(15 downto 0);
111         operand_d2 :out std_logic_vector(15 downto 0);
112         s0re_add,s0im_add: out std_logic;
113         s1re_add,s1im_add: out std_logic;
114         done : out std_logic;
115         clear: out std_logic
116         );
117   end component;
118
119   component add_sub is
120   port(
121        a: in std_logic_vector(15 downto 0);
122         b: in std_logic_vector(15 downto 0);
123         add:in std_logic;
124         ans:out std_logic_vector(15 downto 0)
125         );
126   end component;
127   component mult16
128   port
129      (
130         in1, in2: in std_logic_vector(15 downto 0);
131         Result: out std_logic_vector(15 downto 0)
132      );
133   end component;
134
135   ---------------fin de declaration des composants------
136   begin
137
138   clear_units <= reset or clear_control;
```

```
140  combiner_control_unit: comb_control2 port map(
141
142       clock => clock,
143        reset => reset,
144        rx_re_in=>rx_re_in,
145        rx_im_in=>rx_im_in,
146        h_re_in=>h_re_in,
147        h_im_in=>h_im_in,
148        --
149        operand_a=> op_at,
150        operand_b =>op_bt,
151        operand_c =>op_ct,
152        operand_d =>op_dt,
153        operand_a2=> op_a2t,
154        operand_b2 =>op_b2t,
155        operand_c2 =>op_c2t,
156        operand_d2 =>op_d2t,
157        ---
158        s0re_add=>s0re_op,
159        s0im_add=>s0im_op,
160        s1re_add=>s1re_op,
161        s1im_add=>s1im_op,
162        done=>done_i,
163        clear=>clear_control
164        );
165
166  -- besoin d'entregister les add/sub signaux
167  process (clock)
168  begin
169  if (clock'event and clock ='1') then
170
```

```
 s0re_op_regd <= s0re_op;
 s0im_op_regd <= s0im_op;
 s1re_op_regd <= s1re_op;
 s1im_op_regd <= s1im_op;
 op_a <=op_at;
 op_b <=op_bt;
 op_c <=op_ct;
 op_d <=op_dt;
 op_a2 <=op_a2t;
 op_b2 <=op_b2t;
 op_c2 <=op_c2t;
 op_d2 <=op_d2t;
end if;
end process;
--------
----registre et multiplicateurs
--------

    s0re_reg:process (clock)
   begin
    if (clock'event and clock ='1') then
      if (clear_units='1') then
      s0re_total  <=  x"0000";
     s0re_total2 <=  x"0000";
      else
      s0re_total  <= s0re_sum1;
     s0re_total2 <= s0re_sum2;
      end if;
      end if;
      end process;

```

```
202       s0im_reg:process (clock)
203      begin
204      if rising_edge(clock) then
205       if (clear_units='1') then
206       s0im_total  <=  x"0000";
207      s0im_total2 <=  x"0000";
208       else
209       s0im_total  <= s0im_sum1;
210      s0im_total2 <= s0im_sum2;
211       end if;
212       end if;
213       end process;
214
215      s1re_reg:process (clock)
216      begin
217      if rising_edge(clock) then
218       if (clear_units='1') then
219      s1re_total  <=  x"0000";
220      s1re_total2 <=  x"0000";
221       else
222       s1re_total  <= s1re_sum1;
223      s1re_total2 <= s1re_sum2;
224       end if;
225       end if;
226       end process;
227
228       s1im_reg:process (clock)
229      begin
230      if rising_edge(clock) then
231       if (clear_units='1') then
232       s1im_total  <=  x"0000";
233      s1im_total2 <=  x"0000";
234       else
235       s1im_total  <= s1im_sum1;
236      s1im_total2 <= s1im_sum2;
237       end if;
238       end if;
```

```
239         end process;
240
241         ------------------------------------------------------
242     multip_s0re: mult16 port map
243       (
244         in1=> op_a,
245         in2=> op_b ,
246         Result=>s0re_prod
247       );
248     multip_s0re2: mult16 port map
249       (
250         in1=> op_a2,
251         in2=> op_b2 ,
252         Result=>s0re_prod2
253       );
254    ----------------------------------
255     multip_s0im: mult16 port map
256       (
257         in1=> op_a,
258         in2=> op_c ,
259         Result=>s0im_prod
260       );
261     multip_s0im2: mult16 port map
262       (
263         in1=> op_a2,
264         in2=> op_c2 ,
265         Result=>s0im_prod2
266       );
267    ----------------------------------
268     multip_s1re: mult16 port map
269       (
270         in1=> op_d,
271         in2=> op_b ,
272         Result=>s1re_prod
273       );
```

```
 multip_s1re2: mult16 port map
   (
      in1=> op_d2,
      in2=> op_b2 ,
      Result=>s1re_prod2
   );
------------------------------
 multip_s1im: mult16 port map
   (
      in1=> op_d,
      in2=> op_c ,
      Result=>s1im_prod
   );
 multip_s1im2: mult16 port map
   (
      in1=> op_d2,
      in2=> op_c2 ,
      Result=>s1im_prod2
   );
 -------------------------------
 ---add/sub
 --------------------------------
 s0re_add1: add_sub port map(
   a => s0re_total,
    b => s0re_prod,
    add =>s0re_op_regd,
    ans => s0re_sum1
    );
s0re_add2: add_sub port map(
   a => s0re_total2,
    b => s0re_prod2,
    add =>s0re_op_regd,
    ans => s0re_sum2
    );
```

```
s0re_add: add_sub port map(
   a => s0re_sum1,
    b => s0re_sum2,
    add =>s0re_op_regd,
    ans => s0re_sum
    );
---------------------------------
 s0im_add1: add_sub port map(
   a => s0im_total,
    b => s0im_prod,
    add =>s0im_op_regd,
    ans => s0im_sum1
    );
s0im_add2: add_sub port map(
   a => s0im_total2,
    b => s0im_prod2,
    add =>s0im_op_regd,
    ans => s0im_sum2
    );
s0im_add: add_sub port map(
   a =>s0im_sum1,
    b => s0im_sum2,
    add =>s0re_op_regd,
    ans => s0im_sum
    );
---------------------------------

 s1re_add1: add_sub port map(
   a => s1re_total,
    b => s1re_prod,
    add =>s1re_op_regd,
    ans => s1re_sum1
    );
```

```
s1re_add2: add_sub port map(
   a => s1re_total2,
    b => s1re_prod2,
    add =>s1re_op_regd,
    ans => s1re_sum2
    );
s1re_add: add_sub port map(
   a => s1re_sum1,
    b => s1re_sum2,
    add =>s0re_op_regd,
    ans => s1re_sum
    );
---------------------------------
 s1im_add1: add_sub port map(
   a => s1im_total,
    b => s1im_prod,
    add =>s1im_op_regd,
    ans => s1im_sum1
    );
s1im_add2: add_sub port map(
   a => s1im_total2,
    b => s1im_prod2,
    add =>s1im_op_regd,
    ans => s1im_sum2
    );
s1im_add: add_sub port map(
   a => s1im_sum1,
    b =>  s1im_sum2,
    add =>s0re_op_regd,
    ans => s1im_sum
    );
-----------------------------------------------------
-------------output assignements---------------------
-----------------------------------------------------
process(done_i)
begin
if(done_i='1') then
     s0re_sumt <=  s0re_sum;
      s0im_sumt <=  s0im_sum;
      s1re_sumt <= s1re_sum;
      s1im_sumt <= s1im_sum;
else
     s0re_sumt <=  s0re_sumt;
      s0im_sumt <=  s0im_sumt;
      s1re_sumt <=  s1re_sumt;
      s1im_sumt <=  s1im_sumt;
end if;
end process;
s0re_est <=  s0re_sumt;
s0im_est <=  s0im_sumt;
s1re_est <= s1re_sumt;
s1im_est <= s1im_sumt;
end Behavioral;
```

Mult16 :

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

entity Mult16 is
   port
   (
      in1,in2: in std_logic_vector(15 downto 0);
      Result : out std_logic_vector(15 downto 0)
   );
end entity Mult16;

architecture Behavioral of Mult16 is
signal Result_t:  std_logic_vector(31 downto 0);
begin

   Result_t <= std_logic_vector(signed(in1) * signed(in2));
        Result   <=Result_t(23 downto 8);

end architecture Behavioral;
```

Add/sub :

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

entity add_sub is
port(
     a: in std_logic_vector(15 downto 0);
          b: in std_logic_vector(15 downto 0);
          add:in std_logic;
          ans:out std_logic_vector(15 downto 0)
          );
end add_sub;

architecture Behavioral of add_sub is

begin
    ans <= std_logic_vector(signed(a)+signed(b)) when add='1'
        else
             std_logic_vector(signed(a)-signed(b));
end behavioral;
```

Unité de contrôle :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.std_logic_SIGNED.all;
4
5  library work;
6  use work.mes_types.all;
7
8  entity comb_control2 is
9  port(
10      clock : in std_logic;
11       reset : in std_logic;
12       rx_re_in: in matrix;
13       rx_im_in: in matrix;
14       h_re_in:  in matrix;
15       h_im_in:  in  matrix;
16       --
17       operand_a  :out std_logic_vector(15 downto 0);
18       operand_b  :out std_logic_vector(15 downto 0);
19       operand_c  :out std_logic_vector(15 downto 0);
20       operand_d  :out std_logic_vector(15 downto 0);
21      operand_a2 :out std_logic_vector(15 downto 0);
22       operand_b2 :out std_logic_vector(15 downto 0);
23       operand_c2 :out std_logic_vector(15 downto 0);
24       operand_d2 :out std_logic_vector(15 downto 0);
25       ----------------
26       s0re_add,s0im_add: out std_logic;
27       s1re_add,s1im_add: out std_logic;
28       done : out std_logic;
29       clear: out std_logic
30       );
31
32 end comb_control2;
33
34 architecture Behavioral of comb_control2 is
```

```
35
36  type state_type is (st_rst,st1,st2,st3,st4);
37       signal state, next_state : state_type;
38       signal op_a_i :std_logic_vector(15 downto 0);
39       signal op_b_i :std_logic_vector(15 downto 0);
40       signal op_c_i :std_logic_vector(15 downto 0);
41       signal op_d_i :std_logic_vector(15 downto 0);
42        ------------------------
43       signal s0re_add_i,s0im_add_i:std_logic;
44       signal s1re_add_i,s1im_add_i: std_logic;
45       signal op_a2_i :std_logic_vector(15 downto 0);
46       signal op_b2_i :std_logic_vector(15 downto 0);
47       signal op_c2_i :std_logic_vector(15 downto 0);
48       signal op_d2_i :std_logic_vector(15 downto 0);
49       signal rx_re_reg:  matrix;
50       signal rx_im_reg:  matrix;
51       signal h_re_reg:   matrix;
52       signal h_im_reg:   matrix;
53
54       signal done_i :std_logic;
55       signal clear_i :std_logic;
56  begin
57    input_regs: process (clock,reset)
58     begin
59       if rising_edge(clock) then
60          if (reset='1') then
61            rx_re_reg <=(others =>x"0000");
62            rx_im_reg <=(others =>x"0000");
63            h_re_reg  <=(others =>x"0000");
64            h_im_reg  <=(others =>x"0000");
65              else if (state = st4) or (state= st_rst) then
66               rx_re_reg <=rx_re_in;
67               rx_im_reg <=rx_im_in;
68               h_re_reg  <=h_re_in;
69               h_im_reg  <=h_im_in;
70        end if;
71        end if;
```

```
72        end if;
73        end process;
74    sync_proc: process (clock,reset)
75     begin
76        if rising_edge(clock) then
77           if (reset='1') then
78                 state     <= st_rst;
79                 s0re_add  <= '1';
80                 s0im_add  <= '1';
81                 s1re_add  <= '1';
82                 s1im_add  <= '1';
83
84              operand_a  <= x"0000";
85               operand_b  <= x"0000";
86               operand_c  <= x"0000";
87               operand_d  <= x"0000";
88               operand_a2 <= x"0000";
89               operand_b2 <= x"0000";
90               operand_c2 <= x"0000";
91               operand_d2 <= x"0000";
92               done       <= '0';
93               clear      <= '1';
94    else
95        state      <= next_state;
96        s0re_add   <= s0re_add_i;
97        s0im_add   <= s0im_add_i;
98        s1re_add   <= s1re_add_i ;
99        s1im_add   <= s1im_add_i;
100
```

```
101         operand_a  <= op_a_i;
102         operand_b  <=op_b_i;
103         operand_c  <=op_c_i;
104         operand_d  <=op_d_i;
105         operand_a2 <= op_a2_i;
106         operand_b2 <=op_b2_i;
107         operand_c2 <=op_c2_i;
108         operand_d2 <=op_d2_i;
109          done       <= done_i;
110         clear      <= clear_i;
111
112          end if;
113          end if;
114   end process;
115
116   output_decod: process (state,h_re_reg,h_im_reg,rx_re_reg,rx_im_reg)
117   begin
118   case(state)is
119
120   when st_rst =>
121    op_a_i     <=x"0000";
122    op_b_i     <=x"0000";
123    op_c_i     <=x"0000";
124    op_d_i     <=x"0000";
125    op_a2_i    <=x"0000";
126    op_b2_i    <=x"0000";
127    op_c2_i    <=x"0000";
128    op_d2_i    <=x"0000";
129    --
130    s0re_add_i  <= '1';
131    s0im_add_i  <= '1';
132    s1re_add_i  <= '1';
133    s1im_add_i  <= '1';
134    done_i      <= '0';
135    clear_i     <= '1';
```

```
136
137   when st1 =>
138     op_a_i  <=h_re_reg(0);
139     op_b_i  <=rx_re_reg(0);
140     op_c_i  <=rx_im_reg(0);
141     op_d_i  <=h_re_reg(1);
142     op_a2_i <=h_re_reg(2);
143     op_b2_i <=rx_re_reg(2);
144     op_c2_i <=rx_im_reg(2);
145     op_d2_i <=h_re_reg(3);
146     s0re_add_i <= '1';
147     s0im_add_i <= '1';
148     s1re_add_i <= '1';
149     s1im_add_i <= '1';
150
151     done_i     <= '1';
152     clear_i    <= '1';
153
154   when st2 =>
155     op_a_i  <=h_im_reg(0);
156     op_b_i  <=rx_im_reg(0);
157     op_c_i  <=rx_re_reg(0);
158     op_d_i  <=h_im_reg(1);
159     op_a2_i <=h_im_reg(2);
160     op_b2_i <=rx_im_reg(2);
161     op_c2_i <=rx_re_reg(2);
162     op_d2_i <=h_im_reg(3);
163     s0re_add_i <= '1';
164     s0im_add_i <= '0';
165     s1re_add_i <= '1';
166     s1im_add_i <= '0';
167
168     done_i     <= '0';
169     clear_i    <= '0';
170
```

```
170
171  when st3 =>
172    op_a_i  <=h_re_reg(1);
173    op_b_i  <=rx_re_reg(1);
174    op_c_i  <=rx_im_reg(1);
175    op_d_i  <=h_re_reg(0);
176    op_a2_i <=h_re_reg(3);
177    op_b2_i <=rx_re_reg(3);
178    op_c2_i <=rx_im_reg(3);
179    op_d2_i <=h_re_reg(2);
180    s0re_add_i <= '1';
181    s0im_add_i <= '0';
182    s1re_add_i <= '0';
183    s1im_add_i <= '1';
184
185    done_i  <='0';
186    clear_i <='0';
187
188  when st4 =>
189    op_a_i  <=h_im_reg(1);
190    op_b_i  <=rx_im_reg(1);
191    op_c_i  <=rx_re_reg(1);
192    op_d_i  <=h_im_reg(0);
193    op_a2_i <=h_im_reg(3);
194    op_b2_i <=rx_im_reg(3);
195    op_c2_i <=rx_re_reg(3);
196    op_d2_i <=h_im_reg(2);
197    s0re_add_i <= '1';
198    s0im_add_i <= '1';
199    s1re_add_i <= '0';
200    s1im_add_i <= '0';
201
202    done_i  <='0';
203    clear_i <='0';
204
205  end case;
206
207   end process;
208   process(state)
209    begin
210    next_state <= state;
211    case(state)is
212     when st_rst =>next_state<=st1;
213      when st1 =>next_state<=st2;
214      when st2 =>next_state<=st3;
215      when st3=>next_state<=st4;
216      when st4=>next_state<=st1;
217      when others =>
218      next_state <=st_rst;
219      end case;
220  end process;
221  end Behavioral;
```

Estimateur :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.std_logic_SIGNED.all;
4  use ieee.numeric_std.ALL;
5
6  library work;
7  use work.mes_types.all;
8
9  entity estimateur is
10 port(
11 clock : in std_logic;
12 rx_re_t:in matrix;
13 rx_im_t:in matrix;
14 h_re_t:out matrix;
15 h_im_t:out matrix
16 );
17 end estimateur;
18
19 architecture Behavioral of estimateur is
20      signal c : std_logic_vector(15 downto 0):="0000000100000000";
21      signal c2: std_logic_vector(15 downto 0) := "1111111100000000";
22
23      signal temp_re:matrix;
24       signal temp_im:matrix;
25
26      signal  h_re_prod0re_c :  std_logic_vector(15 downto 0);
27       signal  h_re_prod1re_c2:  std_logic_vector(15 downto 0);
28       signal  h_re_prod1re_c :  std_logic_vector(15 downto 0);
29       signal  h_re_prod2re_c :  std_logic_vector(15 downto 0);
30       signal  h_re_prod3re_c :  std_logic_vector(15 downto 0);
31       signal  h_re_prod3re_c2:  std_logic_vector(15 downto 0);
32       ----------------------------------------------------
```

```
 ----------------------------------------------------
 signal  h_re_prod0im_c2:   std_logic_vector(15 downto 0);
 signal  h_re_prod1im_c2:   std_logic_vector(15 downto 0);
 signal  h_re_prod1im_c :   std_logic_vector(15 downto 0);
 signal  h_re_prod2im_c2:   std_logic_vector(15 downto 0);
 signal  h_re_prod3im_c2:   std_logic_vector(15 downto 0);
signal  h_re_prod3im_c :   std_logic_vector(15 downto 0);
 ----------------------------------------------------
 ----------------------------------------------------------
 signal  h_im_prod0im_c :  std_logic_vector(15 downto 0);
 signal  h_im_prod1im_c2:  std_logic_vector(15 downto 0);
 signal  h_im_prod1im_c :  std_logic_vector(15 downto 0);
 signal  h_im_prod2im_c :  std_logic_vector(15 downto 0);
 signal  h_im_prod3im_c :  std_logic_vector(15 downto 0);
 signal  h_im_prod3im_c2:  std_logic_vector(15 downto 0);
 ----------------------------------------------------
 signal  h_im_prod0re_c2:   std_logic_vector(15 downto 0);
 signal  h_im_prod1re_c2:   std_logic_vector(15 downto 0);
 signal  h_im_prod1re_c :   std_logic_vector(15 downto 0);
 signal  h_im_prod2re_c2:   std_logic_vector(15 downto 0);
 signal  h_im_prod3re_c2:   std_logic_vector(15 downto 0);
signal  h_im_prod3re_c :   std_logic_vector(15 downto 0);

 ----------------------------------------------------------
 ----------------------------------------------------
     signal h_re_sum0_c: std_logic_vector(15 downto 0);
    signal h_re_sum1_c: std_logic_vector(15 downto 0);
    signal h_re_sum2_c: std_logic_vector(15 downto 0);
    signal h_re_sum3_c: std_logic_vector(15 downto 0);

    signal h_re_sum0_c2: std_logic_vector(15 downto 0);
    signal h_re_sum1_c2: std_logic_vector(15 downto 0);
    signal h_re_sum2_c2: std_logic_vector(15 downto 0);
    signal h_re_sum3_c2: std_logic_vector(15 downto 0);

```

```
 signal h_re_sum0: std_logic_vector(15 downto 0);
signal h_re_sum1: std_logic_vector(15 downto 0);
signal h_re_sum2: std_logic_vector(15 downto 0);
signal h_re_sum3: std_logic_vector(15 downto 0);

signal h_im_sum0: std_logic_vector(15 downto 0);
signal h_im_sum1: std_logic_vector(15 downto 0);
signal h_im_sum2: std_logic_vector(15 downto 0);
signal h_im_sum3: std_logic_vector(15 downto 0);

signal h_im_sum0_c2: std_logic_vector(15 downto 0);
signal h_im_sum1_c2: std_logic_vector(15 downto 0);
signal h_im_sum2_c2: std_logic_vector(15 downto 0);
signal h_im_sum3_c2: std_logic_vector(15 downto 0);
signal h_im_sum0_c: std_logic_vector(15 downto 0);
signal h_im_sum1_c: std_logic_vector(15 downto 0);
signal h_im_sum2_c: std_logic_vector(15 downto 0);
signal h_im_sum3_c: std_logic_vector(15 downto 0);

   signal send : std_logic;
  signal compt : integer :=0;

 signal h_im0_c2_v:std_logic_vector (31 downto 0);
signal h_im0_c_v:std_logic_vector (31 downto 0);
signal h_re0_c_v:std_logic_vector (31 downto 0);
signal h_re0_c2_v:std_logic_vector (31 downto 0);
  signal h_re1_c2_v:std_logic_vector (31 downto 0);
signal h_re1_c_v:std_logic_vector (31 downto 0);
signal h_re1_c2_v2:std_logic_vector (31 downto 0);
signal h_re1_c_v2:std_logic_vector (31 downto 0);


signal h_im1_c2_v:std_logic_vector (31 downto 0);
signal h_im1_c2_v2:std_logic_vector (31 downto 0);
signal h_im1_c_v:std_logic_vector (31 downto 0);
signal h_im1_c_v2:std_logic_vector (31 downto 0);

```

```
104
105  signal h_im3_c2_v:std_logic_vector (31 downto 0);
106  signal h_im3_c2_v2:std_logic_vector (31 downto 0);
107
108  signal h_im3_c_v:std_logic_vector (31 downto 0);
109  signal h_im3_c_v2:std_logic_vector (31 downto 0);
110
111  signal h_im2_c2_v:std_logic_vector (31 downto 0);
112  signal h_im2_c_v:std_logic_vector (31 downto 0);
113
114
115  signal h_re3_c_v:std_logic_vector (31 downto 0);
116  signal h_re3_c_v2:std_logic_vector (31 downto 0);
117  signal h_re3_c2_v:std_logic_vector (31 downto 0);
118  signal h_re3_c2_v2:std_logic_vector (31 downto 0);
119  signal h_re2_c_v:std_logic_vector (31 downto 0);
120  signal h_re2_c2_v:std_logic_vector (31 downto 0);
121
122
123
124  ------------------------------------------------------------------
125  component add is
126  port(       a,b     :   in std_logic_vector(15 downto 0);
127              ans  :   out std_logic_vector(15 downto 0));
128  end component;
129  component sub
130  port(
131              a: in std_logic_vector(15 downto 0);
132         b: in std_logic_vector(15 downto 0);
133         ans:out std_logic_vector(15 downto 0)
134        );
135  end component;
```

```
component mult is
   port
   (
      in1, in2: in std_logic_vector(15 downto 0);

      Result: out std_logic_vector(31 downto 0)
   );
end component;
component div is
   port
   (
      in1: in std_logic_vector(31 downto 0);
      Result: out std_logic_vector(15 downto 0)
   );
end component;
-----------------------------------------------FIN DE DECLARATION DES COMPOSANTS---
begin

processh:process(clock)
begin

if rising_edge(clock) then
--if (Compt>9) then
if (Compt>10) then
--   if (Compt>2) then
    Compt <=12;
    send<= '0';
else
 Compt <=Compt+1;
 send <= '1';
 end if;
 end if;
 end process;

```

```
 processus:process(clock,send)
begin
if rising_edge(clock) then
if(send='1') then
 temp_re<=rx_re_t;
 temp_im<=rx_im_t;
 --else
 --temp_re<=(others=>x"0000");
 --temp_im<=(others=>x"0000");
 end if;
 end if;
 end process;
 ---------------------------------------------------------------------------
 multip_re_0re_c: mult port map
   (
      in1=> c,
        in2=> temp_re(0) ,
      Result=>h_re0_c_v
   );
 division_re_0re_c: div port map
   (
      in1=>h_re0_c_v ,
      Result=>h_re_prod0re_c
   );

     multip_re_1re_c2: mult port map
   (
      in1=> c2,
        in2=> temp_re(1) ,
      Result=>h_re1_c2_v
   );
 division_re_1re_c2: div port map
   (
      in1=>h_re1_c2_v ,
      Result=>h_re_prod1re_c2
   );
```

```
207
208  multip_re_2re_c: mult port map
209     (
210       in1=> c,
211         in2=> temp_re(2) ,
212       Result=>h_re2_c_v
213     );
214   division_re_2re_c: div port map
215     (
216       in1=>h_re2_c_v ,
217       Result=>h_re_prod2re_c
218     );
219
220      multip_re_3re_c2: mult port map
221     (
222       in1=> c2,
223         in2=> temp_re(3) ,
224       Result=>h_re3_c2_v
225     );
226   division_re_3re_c2: div port map
227     (
228       in1=>h_re3_c2_v ,
229       Result=>h_re_prod3re_c2
230     );
231  --------------------------------------------------------------------------
232  multip_re_1re_c: mult port map
233     (
234       in1=> c,
235         in2=> temp_re(1) ,
236       Result=>h_re1_c_v
237     );
238   division_re_1re_c: div port map
239     (
240       in1=>h_re1_c_v ,
241       Result=>h_re_prod1re_c
242     );
```

```
multip_re_3re_c: mult port map
   (
      in1=> c,
        in2=> temp_re(3) ,
      Result=>h_re3_c_v
   );
 division_re_3re_c: div port map
   (
      in1=>h_re3_c_v ,
      Result=>h_re_prod3re_c
   );
-------------------------------------------------------------------------
-------------------------------------------------------------------------
multip_re_0im_c2: mult port map
   (
      in1=> c2,
        in2=> temp_im(0) ,
      Result=>h_im0_c2_v
   );
 division_re_0im_c2: div port map
   (
      in1=>h_im0_c2_v ,
      Result=>h_re_prod0im_c2
   );

----------------------------------------
multip_re_1im_c2: mult port map
   (
      in1=> c2,
        in2=> temp_im(1) ,
      Result=>h_im1_c2_v
   );
```

```
 division_re_1im_c2: div port map
   (
     in1=>h_im1_c2_v ,
     Result=>h_re_prod1im_c2
   );
-----------------------------------------
multip_re_1im_c: mult port map
   (
     in1=> c,
       in2=> temp_im(1) ,
     Result=>h_im1_c_v
   );
 division_re_1im_c: div port map
   (
     in1=>h_im1_c_v ,
     Result=>h_re_prod1im_c
   );
-----------------------------------------


multip_re_2im_c2: mult port map
   (
     in1=> c2,
       in2=> temp_im(2) ,
     Result=>h_im2_c2_v
   );
 division_re_2im_c2: div port map
   (
     in1=>h_im2_c2_v,
     Result=>h_re_prod2im_c2
   );
-----------------------------------------

```

```
345   division_im_0im_c: div port map
346     (
347       in1=>h_im0_c_v ,
348       Result=>h_im_prod0im_c
349     );
350
351      multip_im_1im_c2: mult port map
352     (
353       in1=> c2,
354         in2=> temp_im(1) ,
355       Result=>h_im1_c2_v2
356     );
357   division_im_1im_c2: div port map
358     (
359       in1=>h_im1_c2_v2 ,
360       Result=>h_im_prod1im_c2
361     );
362  --------------------------------------------------------------------------
363
364  multip_im_2im_c: mult port map
365     (
366       in1=> c,
367         in2=> temp_im(2) ,
368       Result=>h_im2_c_v
369     );
370   division_im_2im_c: div port map
371     (
372       in1=>h_im2_c_v ,
373       Result=>h_im_prod2im_c
374     );
375
376      multip_im_3im_c2: mult port map
377     (
378       in1=> c2,
379         in2=> temp_im(3) ,
380       Result=>h_im3_c2_v2
381     );
```

```
382   division_im_3im_c2: div port map
383     (
384       in1=>h_im3_c2_v2 ,
385       Result=>h_im_prod3im_c2
386     );
387   ------------------------------------------------------------------------
388   multip_im_1im_c: mult port map
389     (
390       in1=> c,
391         in2=> temp_im(1) ,
392       Result=>h_im1_c_v2
393     );
394    division_im_1im_c: div port map
395     (
396       in1=>h_im1_c_v2 ,
397       Result=>h_im_prod1im_c
398     );
399   ------------------------------------------------------------------------
400   multip_im_3im_c: mult port map
401     (
402       in1=> c,
403         in2=> temp_im(3) ,
404       Result=>h_im3_c_v2
405     );
406    division_im_3im_c: div port map
407     (
408       in1=>h_im3_c_v2 ,
409       Result=>h_im_prod3im_c
410     );
411   ------------------------------------------------------------------------
412   ------------------------------------------------------------------------
```

```
multip_im_0re_c2: mult port map
   (
      in1=> c2,
        in2=> temp_re(0) ,
      Result=>h_re0_c2_v
   );
 division_im_0re_c2: div port map
   (
      in1=>h_re0_c2_v ,
      Result=>h_im_prod0re_c2
   );

-----------------------------------------
multip_im_1re_c2: mult port map
   (
      in1=> c2,
        in2=> temp_re(1) ,
      Result=>h_re1_c2_v2
   );
 division_im_1re_c2: div port map
   (
      in1=>h_re1_c2_v2 ,
      Result=>h_im_prod1re_c2
   );
-----------------------------------------
multip_im_1re_c: mult port map
   (
      in1=> c,
        in2=> temp_re(1) ,
      Result=>h_re1_c_v2
   );
 division_im_1re_c: div port map
   (
      in1=>h_re1_c_v2 ,
      Result=>h_im_prod1re_c
   );
```

```
450
451
452  multip_im_2re_c2: mult port map
453     (
454        in1=> c2,
455          in2=> temp_re(2) ,
456        Result=>h_re2_c2_v
457     );
458   division_im_2re_c2: div port map
459     (
460        in1=>h_re2_c2_v,
461        Result=>h_im_prod2re_c2
462     );
463  ----------------------------------------
464
465
466  multip_im_3re_c2: mult port map
467     (
468        in1=> c2,
469          in2=> temp_re(3) ,
470        Result=>h_re3_c2_v2
471     );
472   division_im_3re_c2: div port map
473     (
474        in1=>h_re3_c2_v2,
475        Result=>h_im_prod3re_c2
476     );
477  multip_im_3re_c: mult port map
478     (
479        in1=> c,
480          in2=> temp_re(3) ,
481        Result=>h_re3_c_v2
482     );
```

```
 division_im_3re_c: div port map
   (
     in1=>h_re3_c_v2,
     Result=>h_im_prod3re_c
   );
 ----------------------------------------
h_re_add0re: add port map(
  a => h_re_prod0re_c,
  b => h_re_prod1re_c2,
    ans => h_re_sum0_c
    );
h_re_add0im: add port map(
  a => h_re_prod0im_c2,
  b => h_re_prod1im_c2,
  ans => h_re_sum0_c2
    );

h_re_sub0: sub port map(
  a => h_re_sum0_c,
  b => h_re_sum0_c2,
  ans => h_re_sum0
    );
----------------------------------------
 h_re_add1re: add port map(
   a => h_re_prod0re_c,
   b => h_re_prod1re_c,
    ans => h_re_sum1_c
    );
 h_re_add1im: add port map(
   a => h_re_prod0im_c2,
   b => h_re_prod1im_c,
   ans => h_re_sum1_c2
    );
```

```
516 h_re_sub1: sub port map(
517    a => h_re_sum1_c,
518    b => h_re_sum1_c2,
519    ans => h_re_sum1
520     );
521 ----------------------------------------
522 h_re_add2re: add port map(
523    a => h_re_prod2re_c,
524    b => h_re_prod3re_c2,
525    ans => h_re_sum2_c
526     );
527 h_re_add2im: add port map(
528    a => h_re_prod2im_c2,
529    b => h_re_prod3im_c2,
530    ans => h_re_sum2_c2
531     );
532 h_re_sub2: sub port map(
533    a => h_re_sum2_c,
534    b => h_re_sum2_c2,
535    ans => h_re_sum2
536     );
537 ----------------------------------------
538 h_re_add3re: add port map(
539   a => h_re_prod2re_c,
540   b => h_re_prod3re_c,
541   ans => h_re_sum3_c
542 );
543 h_re_add3im: add port map(
544   a => h_re_prod2im_c2,
545   b => h_re_prod3im_c,
546 ans => h_re_sum3_c2
547 );
```

```
h_re_sub3: sub port map(
  a => h_re_sum3_c,
  b => h_re_sum3_c2,
  ans => h_re_sum3
);
-------------------------------------------------------------------------------------
-----------------------------------Imaginaire----------------------------------------
 -------------------------------------------------------------------------------------
h_im_add0re: add port map(
  a => h_im_prod0re_c2,
  b => h_im_prod1re_c2,
    ans => h_im_sum0_c2
    );
h_im_add0im: add port map(
  a => h_im_prod0im_c,
  b => h_im_prod1im_c2,
  ans => h_im_sum0_c
    );
h_im_add0: add port map(
  a => h_im_sum0_c2,
  b => h_im_sum0_c,
  ans => h_im_sum0
    );
------------------------------
h_im_add1re: add port map(
   a => h_im_prod0re_c2,
   b => h_im_prod1re_c,
   ans => h_im_sum1_c2
    );
h_im_add1im: add port map(
   a => h_im_prod0im_c,
   b => h_im_prod1im_c,
   ans => h_im_sum1_c
    );
```

```
h_im_add1: add port map(
   a => h_im_sum1_c,
   b => h_im_sum1_c2,
    ans => h_im_sum1
    );
------------------------------
h_im_add2re: add port map(
   a => h_im_prod2re_c2,
   b => h_im_prod3re_c2,
   ans => h_im_sum2_c
    );

h_im_add2im: add port map(
   a => h_im_prod2im_c,
   b => h_im_prod3im_c2,
   ans => h_im_sum2_c2
    );
h_im_add2: add port map(
   a => h_im_sum2_c,
   b => h_im_sum2_c2,
   ans => h_im_sum2
    );
------------------------------
h_im_add3re: add port map(
  a => h_im_prod2re_c2,
  b => h_im_prod3re_c,
  ans => h_im_sum3_c
);

h_im_add3im: add port map(
  a => h_im_prod2im_c,
  b => h_im_prod3im_c,
  ans => h_im_sum3_c2
);
h_im_add3: add port map(
  a => h_im_sum3_c,
  b => h_im_sum3_c2,
ans => h_im_sum3
);
-------------------------------------------------------------------------------------------
-----Sorties-----------------------------------
h_im_t(0) <= h_im_sum0;
h_im_t(1) <= h_im_sum1;
h_im_t(2) <= h_im_sum2;
h_im_t(3) <= h_im_sum3;

h_re_t(0) <= h_re_sum0;
h_re_t(1) <= h_re_sum1;
h_re_t(2) <= h_re_sum2;
h_re_t(3) <= h_re_sum3;

end Behavioral;
```

Démodulateur :

Démodulateur BPSK :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_SIGNED.all;

entity dbpsk is port(
  sre_estd:in std_logic_vector(15 downto 0);
  sim_estd:in std_logic_vector(15 downto 0);
  output : out std_logic
);
end dbpsk;

architecture Behavioral of dbpsk is

begin

output<= not sre_estd(15);

end Behavioral;
```

Démodulateur QAM :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

library work;
use work.mes_types.all;

entity dQAM is port(
  sre_estd :in  std_logic_vector(15 downto 0);
  sim_estd :in  std_logic_vector(15 downto 0);
  sre_estd2:in  std_logic_vector(15 downto 0);
  sim_estd2:in  std_logic_vector(15 downto 0);
  output   :out std_logic_vector(1 downto 0);
  output2  :out std_logic_vector(1 downto 0)
);
end dQAM;

architecture Behavioral of dQAM is
begin

output (0)<= not sim_estd (15);
output (1)<= not sre_estd (15);
output2(0)<= not sim_estd2(15);
output2(1)<= not sre_estd2(15);

end Behavioral;
```

TOP :

TOP BPSK :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_SIGNED.all;
use ieee.numeric_std.ALL;
library work;
use work.mes_types.all;

entity TOP is
port(
clock : in std_logic;
clock1: out std_logic;
reset : in std_logic;
input1 : in std_logic;
input2 : in std_logic;
out1: out std_logic;
out2: out std_logic
);
end TOP;

architecture Behavioral of TOP is
------------------------------------------------------------------------------------------------
-----------------------------------------------DECLARATION DES SIGNAUX--------------------------
------------------------------------------------------------------------------------------------
  signal out_i1 : std_logic_vector(15 downto 0);
  signal out_i2 : std_logic_vector(15 downto 0);
  signal out_q1 : std_logic_vector(15 downto 0);
  signal out_q2 : std_logic_vector(15 downto 0);
------------------------------
--------------------------------
  signal s1_re: std_logic_vector(15 downto 0);
  signal s2_re:std_logic_vector(15 downto 0);
  signal s1_im:std_logic_vector(15 downto 0);
  signal s2_im:std_logic_vector(15 downto 0);
  --------------------------------
  signal h_ret:matrix;
  signal h_imt:matrix;
-------------
```

```
-------------
signal outputi_codeur: matrix;
signal outputq_codeur: matrix;
signal outputi_canal: matrix;
signal outputq_canal:matrix;
-----------------
  signal clk:std_logic;
-------------------------------------------------------------------------------------------------------
----------------------------------declaration des composants-------------------------------------------
-------------------------------------------------------------------------------------------------------
component diviseurf  port(
clock_27Mhz : IN    STD_LOGIC;
clock_27Hz  : OUT   STD_LOGIC
);
end component;

component bpsk port( clk: in std_logic;input :in std_logic;i_out, q_out : out std_logic_vector(15 downto 0));
 end component;

component codeur port (clk : in std_logic;i1_in,i2_in : in std_logic_vector(15 downto 0);
q1_in,q2_in : in std_logic_vector(15 downto 0);
outputq:out matrix;outputi:out matrix
);
end component;

component canal  port(
reset : in std_logic;
clock : in std_logic;
inputi : in matrix;
inputq : in matrix;
outputi: out matrix;
outputq: out matrix

);
end component;


component estimateur port(clock : in std_logic;
rx_re_t:in matrix;
rx_im_t:in matrix;
h_re_t:out matrix;
h_im_t:out matrix
);
end component;

component decodeur
port(clock : in std_logic;reset:in std_logic;rx_re_in:in matrix;rx_im_in:in matrix;h_re_in: in matrix;
h_im_in: in matrix;s0re_est:out std_logic_vector(15 downto 0);s0im_est:out std_logic_vector(15 downto 0);
s1re_est:out std_logic_vector(15 downto 0);s1im_est:out std_logic_vector(15 downto 0));
end component;

component dbpsk  port(
sre_estd:in std_logic_vector(15 downto 0);sim_estd:in std_logic_vector(15 downto 0);
  output : out std_logic
);
end component;

-------------------------------------------------------------------------------------------------------
------------------------FIN DE DECLARATION DES COMPOSANTS----------------------------------------------
-------------------------------------------------------------------------------------------------------
begin
frequence: diviseurf  port map(clock_27Mhz=>clock,clock_27Hz=>clk);
clock1 <=clk;
-----------------------------------------------Modulation--------------------------------------------------------------
bpsk1:bpsk port map(clk=>clk,input=>input1,i_out=> out_i1,q_out=>out_q1);
bpsk2:bpsk port map(clk=>clk,input=>input2,i_out=> out_i2,q_out=>out_q2);

-----------------------------------------------CODAGE-------------------------------------------------------------
le_codeur: codeur port map(clk=>clk,i1_in=>out_i1,i2_in=>out_i2,q1_in=>out_q1,q2_in=>out_q2,outputq=>outputq_codeur,
outputi=>outputi_codeur
);
```

```
-----------------------------------------------Canal-----------------------------------------------------------
Channel: canal port map(reset =>reset,clock=>clk,inputi=>outputi_codeur,inputq =>outputq_codeur,
outputi=>outputi_canal,
outputq=>outputq_canal
);
-----------------------------------------------ESTIMATION-------------------------------------------------------

estimator: estimateur port map(clock=>clk,rx_re_t=>outputi_canal,rx_im_t=>outputq_canal,
h_re_t=>h_ret ,h_im_t=> h_imt );

-----------------------------------------------DECODAGE---------------------------------------------------------

decodor: decodeur port map(clock=>clk,reset=>reset,rx_re_in=>outputi_canal,rx_im_in=>outputq_canal,
h_re_in=>h_ret,h_im_in =>h_imt,s0re_est=>s1_re,s0im_est=>s1_im,s1re_est=>s2_re,s1im_est=>s2_im);

-----------------------------------------------DEMODULATION-----------------------------------------------------

demodulateur1: dbpsk port map(sre_estd=>s1_re,sim_estd=>s1_im,output=>out1);
demodulateur2: dbpsk port map(sre_estd=>s2_re,sim_estd=>s2_im,output=>out2);


end Behavioral;
```

TOP QAM :

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.ALL;
use ieee.std_logic_SIGNED.all;
library work;
use work.mes_types.all;

entity TOPQ is
port(
clock1 : in  std_logic;
reset  : in  std_logic;
input1 : in  std_logic_vector(1 downto 0);
input2 : in  std_logic_vector(1 downto 0);
out1   : out std_logic_vector(1 downto 0);
out2   : out std_logic_vector(1 downto 0)
);
end TOPQ;

architecture Behavioral of TOPQ is
------------------------------------------------------------------------------------------
-----------------------------------------------DECLARATION DES SIGNAUX--------------------
------------------------------------------------------------------------------------------
  signal out_i1 : std_logic_vector(15 downto 0);
  signal out_i2 : std_logic_vector(15 downto 0);
  signal out_q1 : std_logic_vector(15 downto 0);
  signal out_q2 : std_logic_vector(15 downto 0);
  --------------------------------
  signal s1_re  : std_logic_vector(15 downto 0);
  signal s2_re  : std_logic_vector(15 downto 0);
  signal s1_im  : std_logic_vector(15 downto 0);
  signal s2_im  : std_logic_vector(15 downto 0);
  --------------------------------
  signal h_ret  : matrix;
  signal h_imt  : matrix;
  --------------------------------
  signal outputi_codeur: matrix;
  signal outputq_codeur: matrix;
  signal outputi_canal : matrix;
  signal outputq_canal : matrix;
  signal clk :std_logic;
------------------------------------------------------------------------------------------
---------------------------------declaration des composants-------------------------------
------------------------------------------------------------------------------------------
--component diviseurf  port(
--clock_27Mhz   : IN    STD_LOGIC;
--clock_27Hz    : OUT   STD_LOGIC
--);
--end component;
```

```
component QAM_mod port(
input :in std_logic_vector(1 downto 0);
i_out, q_out : out std_logic_vector(15 downto 0));
 end component;

component codeur port (
clk         : in std_logic;
i1_in,i2_in : in std_logic_vector(15 downto 0);
q1_in,q2_in : in std_logic_vector(15 downto 0);
outputq     : out matrix;outputi:out matrix
);
end component;

component canal  port(
reset  : in std_logic;
clock  : in std_logic;
inputi : in matrix;
inputq : in matrix;
outputi: out matrix;
outputq: out matrix
);
end component;

component estimateur port
(clock  :in std_logic;
rx_re_t :in matrix;
rx_im_t :in matrix;
h_re_t  :out matrix;
h_im_t  :out matrix);
end component;

component decodeur
port(clock : in std_logic;
reset   :in std_logic;
rx_re_in:in matrix;
rx_im_in:in matrix;
h_re_in : in matrix;
h_im_in : in matrix;
s0re_est:out std_logic_vector(15 downto 0);
s0im_est:out std_logic_vector(15 downto 0);
s1re_est:out std_logic_vector(15 downto 0);
s1im_est:out std_logic_vector(15 downto 0));
end component;
```

```
component dQAM  port(
  sre_estd : in std_logic_vector(15 downto 0);
  sim_estd : in std_logic_vector(15 downto 0);
  sre_estd2: in std_logic_vector(15 downto 0);
  sim_estd2: in std_logic_vector(15 downto 0);
  output   : out std_logic_vector(1 downto 0);
  output2  : out std_logic_vector(1 downto 0)
);
end component;

------------------------------------------------------------------------------------------------
------------------------FIN DE DECLARATION DES COMPOSANTS---------------------------------------
------------------------------------------------------------------------------------------------
begin
--divf: diviseurf  port map(clock_27Mhz=>clock1,clock_27Hz=>clk);
clk<=clock1;
-----------------------------------------------Modulation---------------------------------------

QAM1:QAM_mod port map(input=>input1,i_out=> out_i1,q_out=>out_q1);
QAM2:QAM_mod port map(input=>input2,i_out=> out_i2,q_out=>out_q2);

-----------------------------------------------CODAGE-------------------------------------------
le_codeur: codeur port map(
clk=>clk,i1_in=>out_i1,i2_in=>out_i2,q1_in=>out_q1,q2_in=>out_q2,
outputq=>outputq_codeur,outputi=>outputi_codeur);

-----------------------------------------------Canal--------------------------------------------
Channel: canal port map(reset =>reset,clock=>clk,inputi=>outputi_codeur,
inputq =>outputq_codeur,outputi=>outputi_canal,outputq=>outputq_canal);
-----------------------------------------------ESTIMATION---------------------------------------

estimator: estimateur port map(clock=>clk,rx_re_t=>outputi_canal,rx_im_t=>outputq_canal,
h_re_t=>h_ret ,h_im_t=> h_imt );

-----------------------------------------------DECODAGE-----------------------------------------
le_decodeur: decodeur port map(clock=>clk,reset=>reset,rx_re_in=>outputi_canal,
rx_im_in=>outputq_canal,h_re_in=>h_ret,h_im_in =>h_imt,s0re_est=>s1_re,
s0im_est=>s1_im,s1re_est=>s2_re,s1im_est=>s2_im);

-----------------------------------------------DEMODULATION-------------------------------------
demodulateur: dQAM port map(
sre_estd=>s1_re,sim_estd=>s1_im,output=>out1,sre_estd2=>s2_re,sim_estd2=>s2_im,output2=>out2);
-----------------------------------------------FIN----------------------------------------------
end Behavioral;
```