République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Electronique

Mémoire de Projet de Fin d'Etudes en vue de l'Obtention du Diplôme d'Ingénieur d'Etat en Electronique

Thème

Implémentation d'un Système OFDM sur une carte FPGA

Proposé et dirigé par
Dr Zidane TERRA

Etudié par
AGUEMOUN AMINA et KACI Damia

Devant le jury composé de
Président : Dr LARBES
Promoteur: Dr Z. TERRA
Examinateur : Mr M.TAGHI

Promotion : juin 2008

Remerciement

Nous tenons d'abord à remercier monsieur TERRA qui nous a donné l'opportunité d'apprendre beaucoup de choses et surtout pour sa bonne humeur, sa gentillesse et sa patience.

Nous remercions le président du jury, Dr LARBES, qui nous a fait l'honneur de présider le jury et d'étudier notre travail, et monsieur TAGHI d'avoir accepté d'examiner notre projet.

Un spécial remerciement à mademoiselle Fathya CHKIRED pour son aide précieuse, son soutien moral et surtout son coup de pouce en VHDL.

Un autre spécial remerciement à monsieur DOUADI avec ses connaissances en XILINX qui nous étaient d'une grande importance.

Amina

A la mémoire de mon grand père djeddou AKLI.

J'aimerais remercier tous ceux que j'aime et qui m'aiment :

Toute ma très chère famille : mon cher PAPA et ma chère MAMAN, mes chers frères KARIM et AKLI, mes chères grands mères manou zineb et manou dahbia, tous mes chers oncles et mes chères tantes, tous mes chers cousins et chères cousines.

Aussi mon très cher MASSINISSA.

Mes amies : SANAA, DAMIA, ZINA, IMENE.

Mes amis : NOUFEL, AMINE, HANI, FOUAD, YACINE.

Et notre SEIGNEUR.

Damia

Je tiens à remercier mes très chères parents, mes quatre sœurs Farah, Drina, Louisa et Feriel qui m'ont toujours soutenu (et supporté !). Mes remerciements vont également à mes chèrs grands parents qui m'accompagnent par leur prières, ainsi que mes tantes et pour leur soutient infini, sans oublier mes amies Yasmine, Amina (pour son écoute et tas d'autre choses), Talia et je n'oublie pas de remercier notre dieu notre créateur celui qui répond à nos prières sans conditionnement.

الملخّص

هذا العمل يقدّم مفهوم تقنية النظام المتعدد الحوامل المتعامدة (OFDM)، مبرمج على مستوى الدارة المبرمجة FPGA. في البداية قمنا بتعريف النظام OFDM، ثم عرجنا بعد ذلك إلى تفصيل الأدوات و البرامج اللازمة لتحقيق هذه البرمجة و هي: ISE وXILINX. في القسم FPGA عرضنا بعض التطبيقات التي تناولت النظام OFDM و برمجته على مستوى الدارة FPGA .أمّا الباب الأخير من هذا البحث فقد تناول عملنا في برمجة النظام OFDM و مراحله التفصيلية.

الكلمات المفاتيح: النظام المتعدد الحوامل المتعامدة (OFDM)، الدارة المبرمجة FPGA، البرنامج ISE و VHDL.

Résumé

Ce travail présente l'implémentation d'un système OFDM sur une carte FPGA VIRTEX II de XILINX. Nous avons tout d'abord abordé la théorie sur l'OFDM, après nous avons présenté les outils de conception à savoir l'ISE et Modelsim. Et enfin nous avons implémenté notre système OFDM. Nous avons effectué une mise au point de l'architecture du système et les étapes nécessaires à sa validation, telles que la synthèse, la simulation, le mapping et à la fin son implémentation.

Mots clés : OFDM, FPGA, VHDL, ISE, Model Sim, Simulation, Synthèse, Implémentation.

Abstract

This work is about the implement of an OFDM system on a FPGA chip which is the VIRTEX II of XILINX. First, we have presented the background around OFDM. Then a presentation on the design on a FPGA Chip, so we have exposed the ISE and ModelSim softwares. At the end of this thesis, we have presented the architecture of an OFDM system to be implemented. In this way, we could synthetise, simulate, map and implement the system above.

Keys words: OFDM, FPGA, ISE, VHDL, Model Sim, Simulation, Synthesis.

Table des matières

Liste des tableaux

Table des figures

Liste des Abbreviations

ADC	Analog to Digital Converter
ADSL	Asymetric Digital Subcriber Line
CDMA	Code Division Multiple Access
COFDM	Coded Orthogonal Frequency Division Multiplexing
CP	Cyclic Prefix
CLB	Configurable Logic Bloc
DAB	Digital Audio Broadcasting
DAC	Digital to Analog Converter
DVB	Digital Video Broadcasting
DFT	Discret Fourier Transform
E/S	Entrée / Sortie
FDM	Frequency Division Multiplexing
FIFO	First Input First Output
FPGA	Field Programmable Gate Array
FFT	Fast Fourier Transform
HDL	Hardware Description Language
ICI	Inter Carier interference
IDFT	Inverse Discret Fourier Transform
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol interference
IOB	Input Output Bloc
I/O	Input /Output
IQ	In phase and Quadrature

IEEE	Institute of Electric and Electronic Engineers
LSB	Least significant bit
LUT	Look Up Table
MHz	Mega Hertz
MSB	Most significant bit
OFDM	Orthogonal Frequency Division Multiplexing
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase and Shift Keying
RAM	Random Access Memory
ROM	Read Only Memory
SRAM	Static Read Only Memory
VHDL	Very High Description Language
WiMAX	Worldwide Interoperability Multiple Access
XST	Xilinx Synthesis Tool
ETSI	European Telecommunications Standards Institute

With the song of your faith ;
you can change the world.
It is never late.....
.....Love is the answer. Morandi

Introduction

Au cours de ces dernières décennies, la transmission numérique a connu une révolution en nouveaux services tel que la télévision et la radio numériques, les réseaux locaux sans fil, l'internet à haut débit et bien sûr la téléphonie mobile. Parallèlement à cette explosion numérique, les gammes spectrales connaissent un engouement important ; ceci a donc motivé la recherche de nombreux schémas de transmission capable de supporter des transmissions à large bande. Les groupes de recherche ont découvert l'intérêt considérable des transmissions multiporteuses orthogonales OFDM (Orthogonal Frequency Division Multiplexing) pour les transmissions à large bande.

Ces nouvelles techniques permettant d'augmenter sensiblement l'efficacité spectrale des systèmes mobiles. L'OFDM est implémenté dans différents standards de réseaux locaux sans fil HiperLAN pour l'Europe, IEEE802.11a pour les Etats-Unis et MMAC au Japon. En effet les systèmes monoporteuses, contrairement à l'OFDM, ne remplissaient pas les conditions de résistance aux trajets multiples et de débit élevé. Aussi d'un point de vue économique les couts de communication sont réduits grâce à la réduction du spectre exploité.

les développements récents de la technologie d'intégration VLSI (Very-Large-Scale-Integration) ont également motivé l'utilisation des systèmes OFDM en rendant leur implémentation commercialement viable. D'autre part, les circuits FPGA (Field Programmable Gate Array), qui sont des circuits programmables standards, et qui peuvent être adaptés à des besoins divers, deviennent incontournables dans les applications nécessitant un temps de développement rapide et une modularité garantie.

L'objet de ce projet est l'implémentation d'un système OFDM sur un circuit FPGA. Ce document se compose de 4 chapitres, qui se répartissent comme suit : le premier chapitre comporte les principes de base du système OFDM, le deuxième chapitre est consacré à la réalisation d'un projet sur circuit FPGA, le troisième chapitre sera consacré à la présentation de la conception des blocs du système OFDM et le quatrième chapitre portera sur les résultats de la simulation, la synthèse et l'implémentation du système OFDM. On terminera par une conclusion générale.

Chapitre 1

OFDM

Si les premières études sur les multiporteuses datent de la fin des années 1950, le multiplex à division de fréquences orthogonales, plus connu sous le nom anglophone $OFDM$ (Orthogonal Frequency Division Multiplexing) a fait son apparition une dizaine d'années plus tard grâce notamment aux travaux de Chang.

Délaissée ensuite lors du développement de la théorie de l'égalisation pour les systèmes onoporteuses (de moindre complexité), l'$OFDM$ dut son retour en grâce vers le milieu des années 1980 au projet de radiodiffusion numérique DAB (Digital Audio Broadcasting). En effet les systèmes monoporteuses, contrairement à l'$OFDM$, ne remplissaient pas les conditions de résistance aux trajets multiples et de débit élevé pour un taux d'erreur binaire faible requis par cette nouvelle application. Depuis lors, l'$OFDM$ est restée une technique prépondérante, puisqu'elle est utilisée pour de nombreuses applications comme la télévision numérique DVB (Digital Video Broadcasting) ou la norme ADSL (Asymetric Digital Suscriber Line) permettant des liaisons internet à haut débit. Enfin l'$OFDM$ s'adapte parfaitement aux communications mobiles, et semble incontournable pour les futurs standards de troisième et quatrième générations.

1.1 Définition

Orthogonal Frequency Division Multiplexing ($OFDM$: Orthogonal Frequency Division Modulation) est une modulation de signaux numériques par répartition en fréquences orthogonales. L'$OFDM$ est un procédé de multiplexage des signaux numériques utilisé entre autres pour les systèmes de transmissions mobiles à haut débit de données. Le multiplexage par répartition orthogonale de la fréquence (Orthogonal Frequency Division Multiplexing ou $OFDM$ est une technique de modulation *multi-porteuses* à base de transformée de Fourier discrète (**1**,). L'$OFDM$ est particulièrement bien adapté aux canaux de transmission radio sans transmissions d'onde multiples. L'idée du multiplexage a pu être dépistée dans le multiplexage fréquentiel classique (FDM), la bande totale est subdivisée en N sous canaux qui ne se chevauchent pas, alors que sur $OFDM$ la bande est divisée en un certain nombre de sous canaux se superposant mais avec des fréquences orthogonales. Le principe de l'$OFDM$ consiste à représenter le signal numérique que l'on veut transmettre par un grand nombre de porteuses (figure 1·1).

Dans ce chapitre, on présentera le système $OFDM$ dans son ensemble : l'émetteur, le canal et récepteur. Cette présentation du système par bloc permettra de réaliser l'archi-

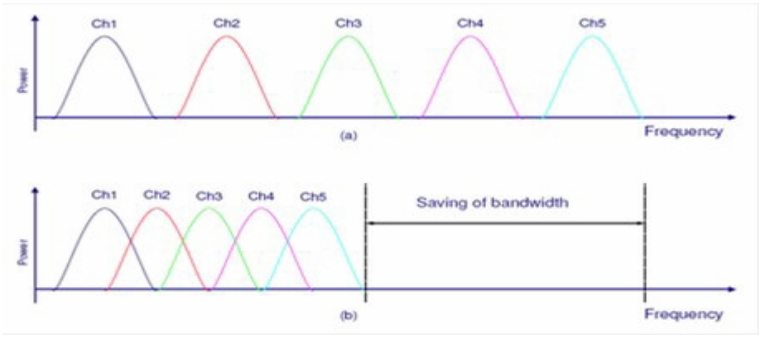

Fig. 1·1: Multiplexage FDM et multiplexage OFDM

tecture du système à implémenter. A la fin du chapitre, nous avons donné un aperçu sur le standard $IEEE$ 802.16 utilisant la technique $OFDM$ ainsi que ses spécifications.

1.2 système multiporteuse

Le principe de la transmission multi-porteuse $OFDM$ (Orthogonal Frequency Division Multiplexing) est de convertir un flux de données série de haut débit en des sous-flux de données parallèles avec un débit faible, et chaque sous-flux module une sous-porteuse différente. Puisque le débit dans chaque sous-porteuse est faible par rapport au débit initial, ainsi l'effet des interférences (ISI) est diminué. (**3**,).

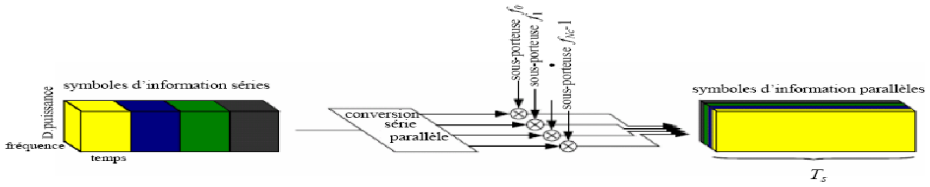

Fig. 1·2: Modulation multi-porteuse avec $N_C = 4$ sous-porteuse

A noter que la représentation à trois dimensions *temps* / *fréquence* / *densité* de puissance, dans la Figure 1·2, est utilisée pour illustrer le principe des systèmes multi-porteuses. Ce principe est un but important dans conception des systèmes de transmission multi-porteuses $OFDM$.

1.3 Notion d'orthogonalité

La différence fondamentale entre les différentes techniques classiques de modulation multiporteuses et l'OFDM est que cette dernière autorise un fort recouvrement spectral entre les porteuses, ce qui permet d'augmenter sensiblement leur nombre ou d'amoindrir l'encombrement spectral. Cependant, pour que ce recouvrement n'ait pas d'effet néfaste, les porteuses doivent respecter une contrainte d'othogonalité, à la fois dans les domaines temporel et fréquentiel.

1.3.1 Orthogonalité temporelle

La contrainte d'orthogonalité temporelle des fonctions $\Psi_{n,k}(t)$ se traduit directement par des propriétés sur le module et l'argument de la forme d'onde $g(t)$ (equation 1.1). La forme d'onde la plus utilisée, parce que la plus simple à générer, est la fonction rectangulaire :

$$g(t) = \left\{ \begin{array}{cc} \frac{1}{\sqrt{T_U}} & sur[0, T_U] \\ 0 & ailleurs \end{array} \right\} \tag{1.1}$$

La fonction porte est la forme d'onde que nous avons retenue pour notre application. Il existe de nombreuses autres formes d'onde acceptables pour la génération de signaux $OFDM$. On peut entre autres citer la fonction cosinus, associée à une modulation MSK, la fonction racine de cosinus surélevé, retenue pour le projet DVB, la fonction de Tuckey ou encore la forme $IOTA$ qui repose sur la théorie des ondelettes.

1.3.2 Orthogonalité fréquentielle

Malgré le recouvrement spectral des porteuses, il doit être possible à tout instant de discriminer deux porteuses voisines pour récupérer les symboles. Cela conduit à un écart inter-fréquentiel minimal, dépendant de la forme d'onde retenue pour la génération de la trame $OFDM$, en dessous duquel la séparation des symboles ne peut plus être assurée.

Afin de réduire au maximum le spectre de fréquence utilisé, la majorité des systèmes employant l'$OFDM$ choisissent comme espace inter-fréquentiel cet écart minimal admissible entre deux porteuses (**4**,).

Dans le cas que nous avons retenu, celui de la fonction porte, il est défini par :

$$\Delta f_{\min} = \frac{1}{T_u} \tag{1.2}$$

L'orthogonalité dans le domaine fréquentiel est réalisée puisque le maximum de chaque sous-porteuse correspond à un ″zéro″ des autres (voir figure 1·3). Cette condition permet ainsi d'avoir une occupation spectrale idéale et d'éviter les interférences entre sous-porteuses.

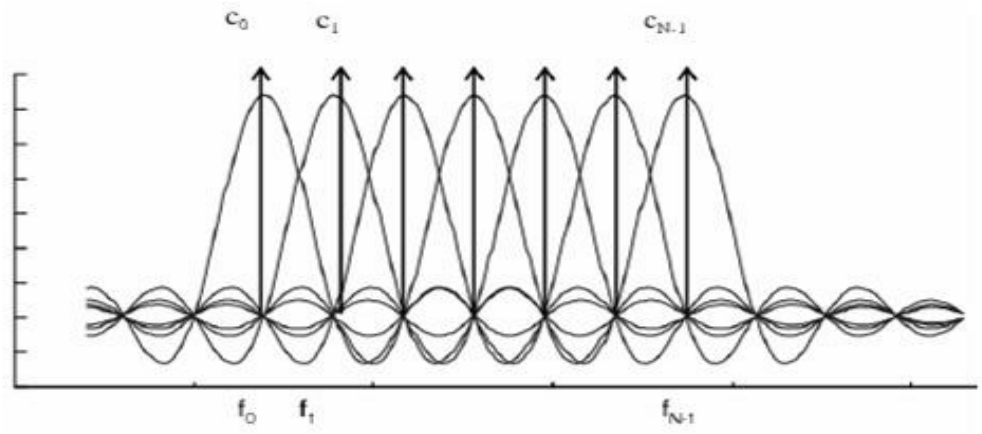

Fig. 1·3: Orthogonalité des sous porteuses

1.4 ISI , ICI et le préfixe cyclique

Les interférences entre symboles (ISI) sont dues au comportement multi-trajets du canal, le signal reçu provenant de la contribution du trajet direct et des trajets multiples introduisant des déphasages et des retards, ces derniers pouvant être du même ordre de grandeur que la durée d'un symbole (**28**,). Pour remédier à ce problème, on ajoute entre deux trames $OFDM$, un intervalle de garde dont la durée doit être supérieure au retard maximum des signaux issus des trajets indirects (voir figure 1·4).

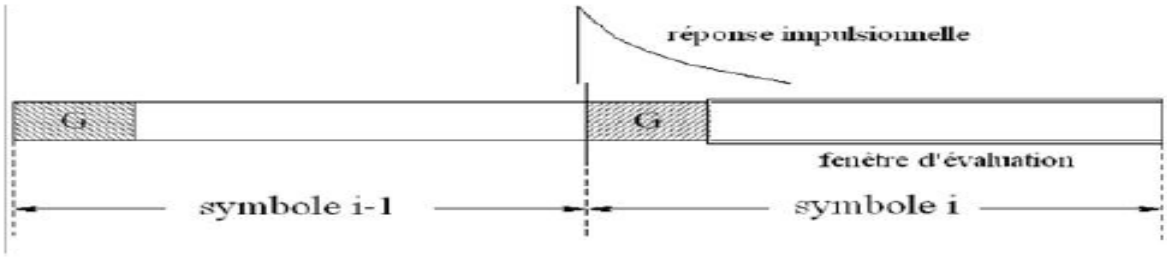

Fig. 1·4: Interférence entre le symbole i et i-1

Un des problèmes majeurs que rencontre l'$OFDM$ est sa sensibilité à la difference des fréquences entre l'émetteur et le récepteur appelée la fréquence offset qui est dûe au décalage de fréquence ou effet Doppler dans le canal. Cette fréquence cause la perte de l'orthogonalité entre les sous porteuses dont les passages par zéro se produiront donc pour des valeurs différentes de celles associées au trajet direct. Lors de l'échantillonnage, il

n'y aura plus d'orthogonalité entre les sous-porteuses et on retrouvera des informations d'une sous porteuse sur l'autre(ICI). Afin d'éviter ces interférences, on doit ajouter ce qu'on appelle un intervalle de garde qui n'est d'autre que la copie des L derniers symboles d'information du symbole $OFDM$. On parle dans ce cas de préfixe cyclique (que l'on développera dans la section prochaine).

1.5 Système OFDM

On commencera d'abord par donner le processus de transmission théoriquement dans un schéma de principe d'un système OFDM (voir figure 1·5).

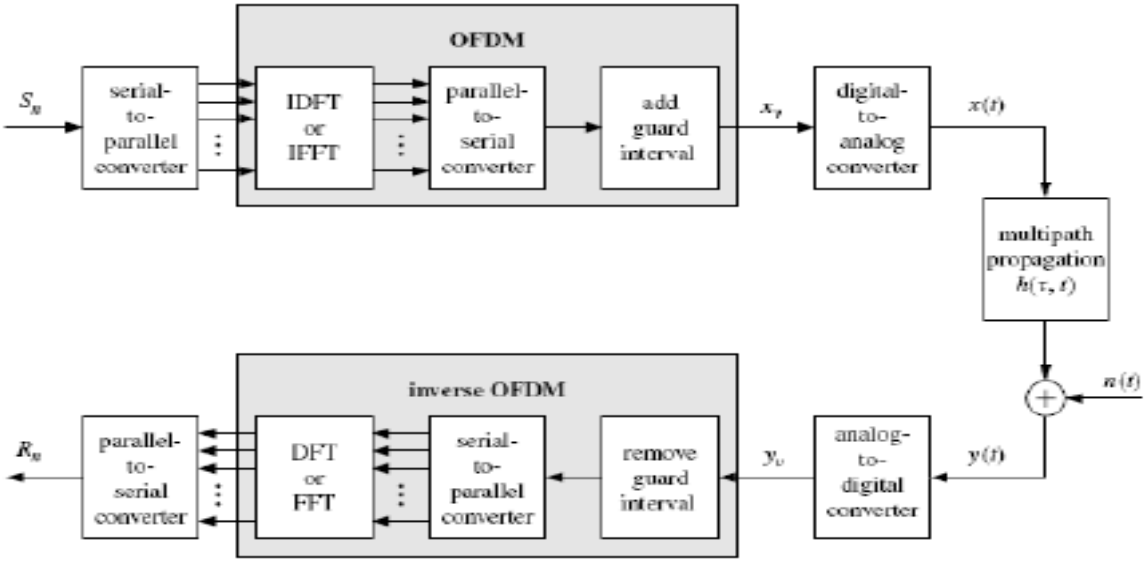

Fig. 1·5: Système de transmission OFDM

Un système de communication qui utilise la modulation multi-porteuse, transmis N_c symboles source de valeurs complexes $S_n, n = 0, ..., N_c - 1$ en parallèle dans $N_c$sous porteuses différentes (**13**,). Les symboles sources sont en général obtenus après codage de source et de canal, entrelacement et opération de *mapping*. Après la conversion *série-parallèle*, la durée d'un symbole $OFDM$ sera :

$$N_s = N_c T_d \tag{1.3}$$

où : T_d est la durée d'un symbole d'information.

L'espacement entre les Nc sous-porteuses est donné par : $F_s = \dfrac{1}{T_s}$

Les N_c symboles modulés sont transmis comme un symbole $OFDM$ avec une enveloppe

complexe donnée par :

$$x(t) = \frac{1}{N_c} \sum_{n=0}^{N_c-1} S_n e^{j2\pi f_n t} \quad 0 \leq t \leq T_S \text{ où } \quad f_n = \frac{n}{T_S} \quad n = 0, ...N_C - 1 \qquad (1.4)$$

Le premier avantage dans l'utilisation de l'$OFDM$ est que la modulation *multi-porteuse* peut être facilement implémentée dans le domaine discret en utilisant $IDFT$ ou $IFFT$. L'$IDFT$ qui a comme coefficients la séquence des symboles d'information, n'est rien d'autre que l'échantillonnage de l'enveloppe complexe $x(t)$ à la fréquence $\frac{1}{T_d}$. Elle est donnée par :

$$x_v = \frac{1}{N_c} \sum_{n=0}^{N_c-1} S_n e^{\frac{j2\pi NV}{N_c}} \quad v = 0, ..., N_c - 1 \qquad (1.5)$$

Cependant pour éviter complètement l'ISI et ICI, on ajoute un intervalle de garde entre les symboles $OFDM$ adjacents. Cet intervalle doit être d'une durée :

$$T_g \geq \tau_{\max}$$

$\tau_{\max}$ est le retard maximal d'un trajet indirect. La durée d'un $OFDM$ symbole devient :

$$\acute{T_S} = T_S + T_g \qquad (1.6)$$

La longueur discrète de l'intervalle de garde est L_g . Donc la valeur de L_g doit vérifier :

$$L_g \geq \left[\frac{\tau_{\max} N_C}{T_S} \right]$$

La séquence échantillonnée avec intervalle de garde devient :

$$x_v = \frac{1}{N_c} \sum_{n=0}^{N_c-1} S_n e^{\frac{j2\pi NV}{N_c}} \quad v = -L_g, ..., N_C - 1 \qquad (1.7)$$

Cette séquence va passer à travers un convertisseur numérique/analogique dont la sortie est un signal de forme d'onde $x(t)$, de durée $T_S^{'}$, qui sera transmis à travers le canal radio mobile. La sortie du canal est obtenue après convolution de $x(t)$ avec la réponse impulsionnelle $h(\tau, t)$ et l'addition du bruit $n(t)$:

$$y(t) = \int_{-\infty}^{+\infty} x(t-\tau) h(\tau, t) d\tau + n(t) \qquad (1.8)$$

Le signal reçu $y(t)$ va passer à travers un convertisseur analogique/numérique, dont la sortie est la séquence $y_v, v = -L_g, ..., N_C - 1$, qui est l'échantillonnage du $y(t)$ à la fréquence $\frac{1}{T_d}$. Puisque l'ISI existe seulement dans les L_g premiers échantillons de la séquence reçue, ces échantillons sont enlevés avant une démodulation multi-porteuse.

La partie des échantillons y_v libre d'$ISI (v = 0, ..., N_C - 1)$ est démodulée par inverse $OFDM$ en utilisant l'FFT. La séquence démodulée R_n est donnée par :

$$R_n = \sum_{v=0}^{N_c-1} y_v e^{\frac{j2\pi nv}{N_C}} \quad n = 0, ..., N_C - 1 \tag{1.9}$$

1.6 Etude d'un Emetteur OFDM

À l'émetteur, une conversion série/parallèle est appliquée à la séquence d'information en série divisée en blocs. Après, chaque bloc est mappé à travers une constellation. Ensuite, une modulation aux M sous porteuses parallèles dans le domaine fréquentiel (**6**,). Les valeurs complexes résultantes sont converties par une $IFFT$ de taille $N \geq M$ du domaine fréquentiel vers le domaine temporel. Le préfixe cyclique (ou l'intervalle de garde) est alors ajouté à chaque symbole avant la conversion numériques-analogique et puis la transmission.

1.6.1 Conversion série-parallèle

Les données à transmettre sont sous forme d'une séquence d'information série (**31**,). La conversion consiste à convertir cette dernière en sous séquences parallèles. Cette étape est nécessaire pour transmettre un nombre important d'information par un seul symbole $OFDM$. Le nombre de bits transmis dans chaque symbole $OFDM$ dépend des schémas de modulations utilisées et du nombre des sous porteuses utilisées.

1.6.2 Mapping

On subdivise la séquence d'information en sous blocs de R bits. On utilisera une modulation M-aire M QAM (M-ary Quadrature Amplitude Modulation). La taille M de la constellation est donnée par $M = 2^R$. Chaque R bits est représenté par un signal correspondant à un point de la constellation. Ce point peut être représenté par un nombre complexe dont les parties réelles et imaginaires sont notées respectivement I et Q. La Figure 1·6 montre 3 constellations : 4-QAM, 16-QAM et la 64-QAM. On utilise un codage de Gray dans l'attribution des points signaux ou R bits.

Chaque constellation est bâtie sur le même modèle, à savoir que les parties réelles et imaginaires des symboles prennent leurs valeurs dans l'ensemble $\{\pm v, \pm 3v, \pm 5v, \ldots\}$ (**22**,).

1.6.3 L'IFFT

Après l'étape du mapping au domaine fréquentiel, avant l'application de l'$IFFT$**,** chaque échantillon de l'IFFT correspond à une seule sous porteuse. La plupart des sous porteuses sont modulées par les données. Les sous porteuses périphériques ne sont pas

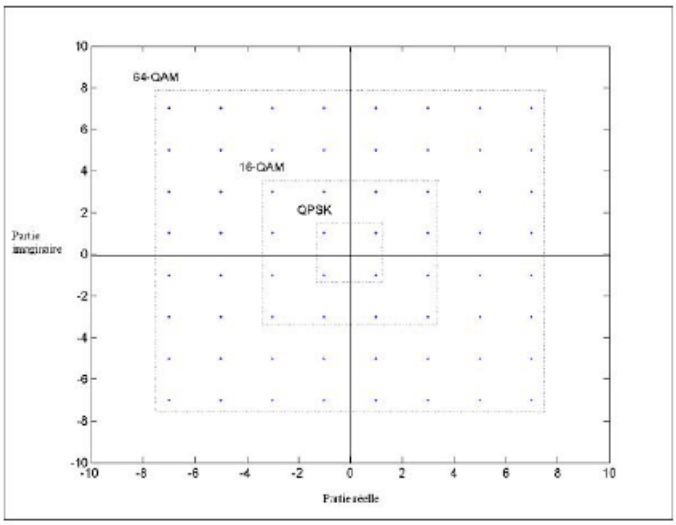

Fig. 1·6: Constellations 4QAM, 16QAM et 64QAM

modulées, c'est-à-dire elles sont mises à zéro. L'$IFFT$ est utilisé pour convertir ce signal au domaine temporel. Les N symboles signaux $c_0, c_1, ..., c_{N-1}$ constituent un symbole $OFDM$. Le $k^{iéme}$ symbole signal module une sous porteuse de fréquence f_k. La fréquence,

en bande de base, de chaque sous porteuse est un multiple de l'inverse de la durée du symbole $OFDM$, ce qui implique que chaque sous porteuse a un nombre entier de période par symbole $OFDM$. Cette propriété entraîne la vérification de la condition d'orthogonalité entre les sous porteuses. Ainsi, lorsque l'échantillonnage est effectué précisément à la

fréquence f_k d'une sous-porteuse, il n'y a aucune interférence avec les autre sous-porteuses. Ceci permet d'obtenir ainsi une occupation optimale du spectre.

1.6.4 Insersion de l'Intervalle de garde

Le préfixe est constitué des ν derniers échantillons $[x(N-\nu), ..., x(N-1)]$ du symbole $OFDM$ (voir figure 1·7). Ainsi, les différentes répliques du symbole ont un nombre entier de cycles dans l'intervalle de l'$IFFT$ et les sous-porteuses n'interfèrent plus (**29**,).

L'ajout de ce temps de garde T_g a pour but d'éliminer l'ISI qui subsiste malgré l'orthogonalité des porteuses. Pour que cet intervalle de garde soit efficace, sa durée doit être au moins égale à l'écho non négligeable le plus long (celui qui a le retard maximal) des canaux sélectifs en fréquence.

Entre la période symbole, la période utile et l'intervalle de garde s'instaurent donc la relation (equation 1.10) :

$$T_S = T_U + T_g \tag{1.10}$$

Fig. 1·7: Principe du préfixe cyclique

où T_g est la durée de l'intervalle de garde ajouté et T_U est la durée initiale du symbole généré par l'IFFT. En plus de la protection du signal OFDM contre l'ISI, l'intervalle de garde assure également la protection contre les effets du décalage temporel entre le récepteur et l'émetteur.

1.6.5 Modulation Radio Fréquence

L'autre possibilité est d'effectuer une transposition en fréquence, permettant ainsi de moduler respectivement une porteuse en phase et une porteuse en quadrature, par les parties réelle et imaginaire de la trame $OFDM$ (voir figure 1·8) .

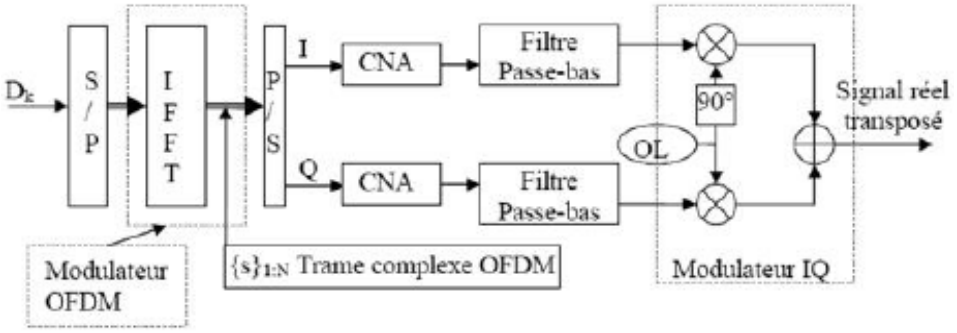

Fig. 1·8: Schéma de modulation ou transposition fréquentielle dans un système OFDM

A la sortie du module assurant l'$IFFT$, on sépare les parties réelles I et imaginaires Q des signaux $OFDM$, et on effectue ensuite une conversion parallèle série. Ces données numériques sont converties en données analogiques (CNA) et ensuite traversent un filtre passe bas. Le spectre $OFDM$ est transposé autour de la fréquence de l'oscillateur local par le modulateur (I, Q). La modulation radio fréquence (RF) est donc implémentée grâce à des techniques analogiques classiques (**28**,).

1.7 Récépteur OFDM

Au récepteur, après conversion analogique-numérique et suppression du préfixe cyclique, une FFT de taille N agit pour traduire le symbole OFDM reçu en des signaux d'information de données complexes. Les opérations de décodage et de demapping sont utilisées pour récupérer le flux des données transmises.

On suppose que le signal est synchronisé, ce qui implique que le début de chaque paquet et la fréquence d'échantillonnage sont corrects.

1.7.1 La suppression de l'intervalle de garde

Cette étape consiste à supprimer la copie de la fin du symbole qui était ajouté à l'emission pour éliminer les interférences entre symboles.

1.7.2 FFT

Le signal parvenu au bloc de la FFT après suppression de l'intervalle de garde, ce bloc convertit le symbole OFDM en des signaux d'information. Il réalise l'opération inverse de l'IFFT.

1.7.3 Le demapping

Le demapping n'est que l'opération inverse du Mapping c'est à dire correspondre à un vecteur (I,Q) un ensemble de bits selon le map ou la constellation utilisée à l'émission.

1.7.4 La notion de Synchronisation

Avant de pouvoir démoduler les sous-porteuses, un récepteur doit effectuer deux synchronisations. D'une part, il doit déterminer les limites du signal et l'instant optimal d'échantillonnage afin de minimiser les effets de l'interférence entre symboles et de l'interférence entre porteuses. D'autre part, il doit estimer et corriger le décalage fréquentiel (offset) responsable d'interférence entre porteuses. Ce décalage fréquentiel résulte de la différence entre les fréquences des oscillateurs de l'émetteur et celui du récepteur.

Sensibilité aux offsets fréquentiels

Les décalages fréquentiels résultent des différences entre les fréquences des oscillateurs de l'émetteur et du récepteur. Deux effets destructifs sont engendrés par un offset fréquentiel. L'un consiste en la réduction de l'amplitude du signal (les fonctions sinc sont décalées et ne sont donc plus échantillonnées aux pics), l'autre est l'introduction d'interférence entre les porteuses résultant de la perte d'orthogonalité entre les sous-canaux. Les systèmes multi-porteuses sont beaucoup plus sensibles aux décalages fréquentiels que les systèmes à porteuse unique (**3**,). La figure 1·9 représente le décalage fréquentiel :

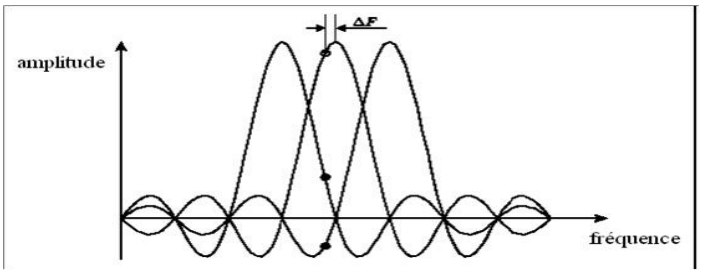

Fig. 1·9: Effets d'un offset fréquentiel δf

δf est l'offset fréquentiel normalisé par l'écart entre porteuses. La sensibilité est donc d'autant plus importante que les porteuses sont proches.

Sensibilité aux erreurs de timing

Dans le paragraphe précédent on a expliqué la grande sensibilité de l'$OFDM$ par rapport aux offsets fréquentiels. Comparativement, l'$OFDM$ est plus robuste vis-à-vis des erreurs de timing. En effet, l'offset temporel peut varier sur un intervalle équivalent au temps de garde sans engendrer d'ICI ou d'ISI. Ces deux types d'interférences apparaissant seulement quand la fenêtre FFT s'étend sur deux symboles, c'est à dire.

$$t_{offset} < 0 \ ou \ t_{offset} > t_{garde}$$

Cependant, pour avoir la meilleure résistance possible face aux canaux multitrajets, il existe un instant d'échantillonnage optimal, cet instant correspond au début de la fenêtre contenant la plus grande énergie parmi toutes les fenêtres de taille ν sur la séquence reçue. Toute déviation par rapport à cet instant est responsable d'une plus grande sensibilité à l'étalement de délai du canal. Pour minimiser la perte de robustesse, le système doit être conçu de telle façon que l'erreur de timing soit petite devant le temps de garde.

1.8 WiMAX IEEE802.16d

Actuellement, la technique $OFDM$ est utilisée dans diverses applications :
- transmission sur câble (HDSL, ADSL, VDSL,etc...),
- diffusion numérique
- les réseaux de communication sans fil tel que les réseaux WLAN (Wireless Local Area Network) pour les standards IEEE802.11a, IEEE802.11g et HiperLAN2 dans la gamme de 5GHz ;
- accès à internet à haut débit sans fil : les réseaux WMAN (Wireless Metropolitan Area Network) pour le standard IEEE802.16 etc....
- etc...

Notre choix du standard IEEE 802.16d a été basé sur les différents éléments d'appréciations qu'offre cette norme, on développera ceci dans le paragraphe qui suit.

Avec les acquis du **WiFi**, de nouveaux besoins naissent. D'une part une demande de sécurité puisque les algorithmes de codages utilisés dans le **WiFi** sont devenus inefficaces (WEP, WPA,. . .). D'autre part, les utilisations actuelles demandent également une garantie de la qualité de service pour pouvoir assurer des services comme de la diffusion multimédia. L'ensemble de ces caractéristiques sera prise en compte dans le **WiMax**.

Une différence majeure entre le **WiFi** et le **WiMax** est que le **WiMax** est conçu pour couvrir de longues distances avec un débit élevé. De plus, une version « mobile » est prévue pour le **WiMax**, celle-ci donnera à l'utilisateur en mouvement les mêmes avantages qu'un utilisateur fixe. A l'échelle économique le développement de cette norme permet de faire baisser le coût des équipements, assurer l'interopérabilité et de réduire le risque d'investissement pour les opérateurs. C'est la première norme industrywide qui peut être utilisé pour l'accès sans fil avec une largeur de bande plus élevée que celle des réseaux cellulaires mobiles.

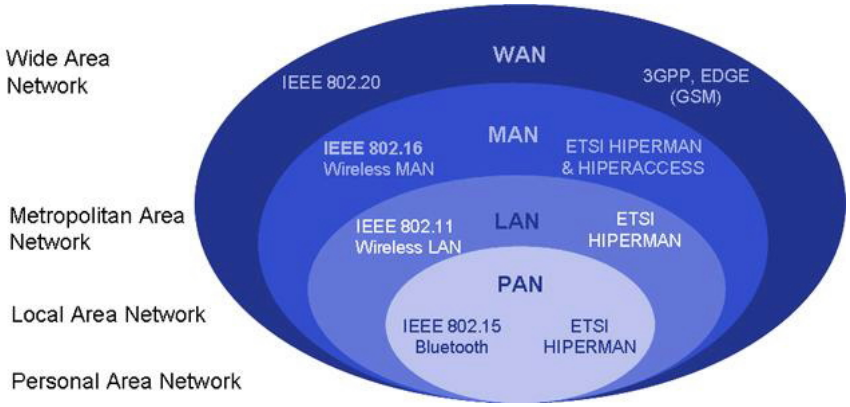

Fig. 1·10: L'étendu du WiMAX

Le schéma de la figure 1·10 permet de distinguer les différentes technologies qui sont utilisées en fonction du type de réseau. La version de la norme $IEEE802,16$ opère dans le $10-66Ghz$ bande de fréquence.

WiMAX Forum a adopté deux versions de la norme $IEEE$ 802.16 pour fournir différentes types d'accès :

- Fixe / accès nomades : Le forum **WiMAX** a adopté $IEEE802.16(2004)$ et $HyperMAN$ (norme d'ETSI) pour l'accès fixes et nomades. Celui-ci utilise Orthogonal Frequency Division Multiplexing et est en mesure de fournir des supports dans un environnement LOS et $NLOS$. Les deux l'extérieur et l'intérieur centres de la petite enfance sont disponibles pour l'accès fixe. Le Forum **WiMAX** a normalisé bande de fréquences à $3,5\ Ghz$ et $5,8\ Ghz$.
- Portable / Mobile Access : Le forum a adopté la norme $IEEE$ 802.16e qui a été optimisé pour les canaux de radio mobile. Cette fonction utilise Scalable $OFDM$

access qui fournit un appui pour handoffs et d'itinérance [17]. Un réseau se basant sur le standard $IEEE$ 802.16e est aussi capable de fournir un accès fixe. Le **WiMAX** mobile utilise les fréquences $5, 7, 8, 75$ et $10Mhz$ sous licence de la fédération des allocations de spectre dans les bandes de fréquences de 2.3 Ghz, 2.5 Ghz, 3.3 Ghz et 3.5 Ghz [18]. Le premier produit certifié est disponible depuis la fin de 2007.

1.9 Conclusion

Nous avons présenté la modulation par répartition orthogonale de fréquence, qui pourrait être considérée comme technique de modulation ou d'accès multiple. Les caractéristiques principales qui définissent un système $OFDM$ sont :

- le nombre de sous porteuses,
- largeur de bande, durée de symbole, l'intervalle de garde et type de modulation.

D'autres paramètres relatifs qui sont : le nombre et la position des pilotes de porteuses. Nous avons présenté la partie du récepteur avec l'Influence de la synchronisation sur les performances du système. Enfin nous avons présenté le standard 802.16 de WIMAX, un standard que nous avons choisi pour les différents avantages cités dans le chapitre.

Chapitre 2

FPGA, XILINX ISE et Model Sim

2.1 FPGA

Les circuits $FPGA$ (Field Programmable Gate Array) sont certainement les circuits reprogrammables ayant le plus de succès. Ce sont des circuits entièrement configurables par programmation qui permettent d'implanter physiquement, par simple programmation, n'importe quelle fonction logique.

Actuellement deux Leaders mondiaux se disputent le marché, *Altéra* et $Xillinx$. De nombreux autres fabricants de moindre envergure, proposent également leurs propres produits (**27**,).

L'objet de notre étude est d'implémenter le système $OFDM$ déjà cité dans le chapitre I sur un circuit $FPGA$. La programmation a été faite en utilisant le langage de description $VHDL$. Nous allons d'abord faire une description des circuits $FPGA$, ensuite on va introduire les étapes nécessaires au développement d'un projet sur un circuit $FPGA$, de la programmation jusqu'au chargement sur la carte, et on terminera par le langage $VHDL$. Nous nous intéresserons aux derniers circuits présents sur le marché à savoir le circuit $FPGA$ de la famille $Virtex - II$. Cette dernière présente une densité d'intégration de $[4à10millionsdeportes]$, avec une vitesse dépassant les $420Mhz$.

Les $FPGA$ sont utilisés dans de nombreuses applications, on en cite dans ce qui suit quelques unes :

- prototypage de nouveaux circuits ;
- fabrication de composants spéciaux en petite série ;
- adaptation aux besoins rencontrés lors de l'utilisation ;
- systèmes de commande à temps réel ;
- DSP (Digital Signal Processor) ;
- imagerie médicale.

2.1.1 Architecture retenue par Xilinx

Les circuits $FPGA$ possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrées/sorties et des blocs logiques programmables (figure 2·1). Le passage d'un bloc logique à un autre se fait par un routage programmable. Certains circuits

$FPGA$ intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeurs (**27**,).

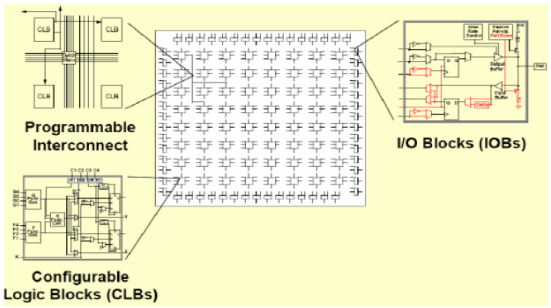

Fig. 2·1: Architecture interne du FPGA

Dans ce qui suit, on va faire une description de l'architecture utilisée par $Xilinx$, car c'est sur des circuits $Xilinx$ qu'on va implémenter nos programmes. L'architecture retenue par $Xilinx$ se présente sous forme de deux couches :

- Une couche appelée Circuit Configurable,
- Une couche réseau mémoire $SRAM$ (Static Read Only Memory).

2.1.2 Circuit configurable

La couche dite « circuit configurable » est constituée d'une matrice de blocs logiques configurables (CLB) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles (figure 2·2).

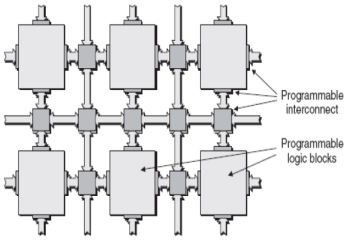

Fig. 2·2: Blocs et Interconnections programables

Tout autour de ces blocs logiques configurables, on trouve des blocs d'entrées/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs

2.1.3 Réseau mémoire SRAM

La programmation du circuit $FPGA$, appelé aussi LCA (Logic Cells Arrays), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux (**9**,). Ces potentiels sont tout simplement mémorisés dans le réseau mémoire $SRAM$ (figure 2·3).

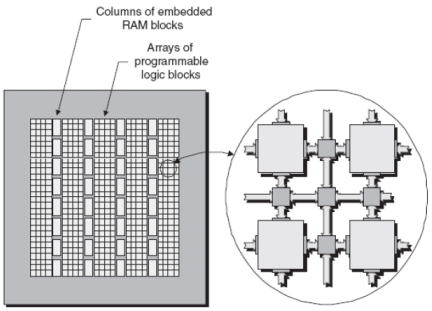

Fig. 2·3: Situation du réseau SRAM

La programmation d'un circuit $FPGA$ est volatile, la configuration du circuit est donc mémorisée sur la couche réseau $SRAM$ et stockée dans une ROM externe. Un dispositif interne permet à chaque mise sous tension de charger la $SRAM$ interne Figure à partir de la ROM. Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive. On voit ici tout le profit que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée.

Les circuits $FPGA$ du fabricant $Xilinx$ utilisent deux types de cellules de base : les cellules d'entrées/sorties appelés IOB (Input Output Bloc), et les cellules logiques appelées CLB (Configurable Logic Bloc). Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurable. On décrit dans ce qui suit chacun de ces composants.

2.1.4 CLB (Configurable Logic Bloc)

Les blocs logiques configurables sont les éléments déterminants des performances du circuit $FPGA$ (figure 2·4). Chaque CLB est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées (LUT) et d'un bloc de mémorisation/synchronisation composé de bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB (**10**,).

La LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc $2^4 = 16$combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient

ainsi un petit bloc générateur de fonctions. La Figure montre le schéma simplifié d'un CLB de la famille $XC4000deXilinx$. Dans cette famille, la cellule de base contient deux $LUTs$ à 4 entrées qui peuvent réaliser deux fonctions quelconques à 4 entrées. Une troisième LUT peut réaliser une fonction quelconque à 3 entrées à partir des sorties des deux premières LUT (F' et G' qui deviennent H2 et H3) et d'une troisième variable d'entrée H1 sortant du bloc « sélecteur ». Le bloc sélecteur contient 4 signaux de contrôle : 3 signaux dédiées pour les registres : une donnée "Din", un signal de validation "Ec" et une remise à un ou à zéro asynchrone "S/R", et le $4^{iéme}$signal représente l'entrée $H1$de la LUT à 3 entrées.

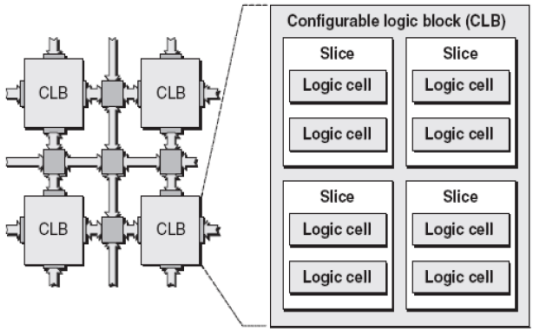

Fig. 2·4: Bloc CLB

$Xilinx$ propose également des composants à "haute densité" avec les familles $Virtex$ (4 millions de portes) et $VirtexII$ (6 millions de portes). La famille $Virtex$ apporte plusieurs nouveautés par rapport à la famille $XC4000$ (figure 2·5) :

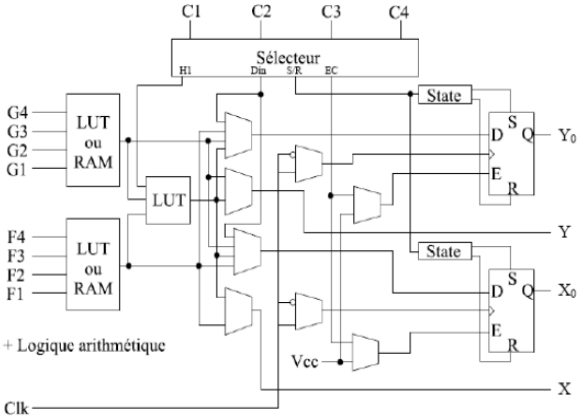

Fig. 2·5: Schéma d'une cellule logique (XC4000 de Xilinx)

– l'adjonction de blocs mémoires de $4Kbits$ au coeur de la logique et même de plus

larges mémoires dans la famille "Extended Memory".
- l'utilisation de boucles à verrouillage de phase améliorées : *DLL* (Digital Delay Locked Loop) qui permettent de synchroniser une horloge interne sur une horloge externe, de travailler en quadrature de phase et de multiplier ou diviser la fréquence.
- La présence d'un anneau de connexions autour du circuit pour faciliter le routage des entréessorties.
- La compatibilité avec de nombreux standards de transmission de données et de niveaux logiques.

2.1.5 IOB (Input Output Bloc)

Ces blocs d'entrée/sortie permettent l'interface entre les broches du composant *FPGA* et la logique interne développée à l'intérieur du composant (figure 2·6).

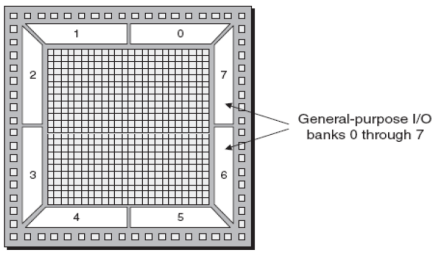

Fig. 2·6: Exemple de IOB

Ils sont présents sur toute la périphérie du circuit *FPGA*. Chaque bloc *IOB* contrôle une broche du composant et il peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance). La figure 2·7 présente la structure de ces blocs.

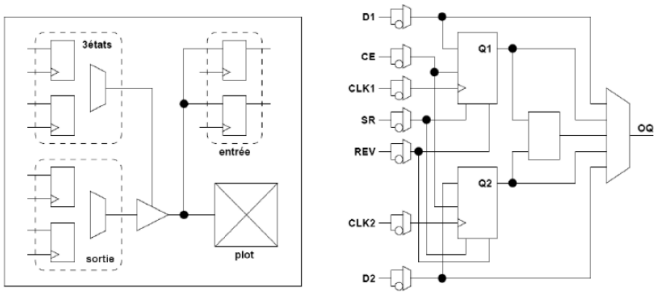

Fig. 2·7: Schéma d'un bloc d'entrée/sortie (IOB)

2.2 Logiciel de développement ISE (Integrated Software Environment)

Dans cette section, nous donnons une description sur notre logiciel de simulation $ISE8.2i$. L'objectif visé est de présenter les différents outils disponibles dans l'ISE et la manière de les intégrés ou de les utilisés dans une conception à l'aide d'un dispositif de $Xilinx$. ISE contrôle toutes les opérations nécessaires pour la conception. A travers son interface appelé navigateur de projet, on peut accéder à tous les outils susceptibles d'intégré la synthèse, la simulation ou l'implémentation. On peut également accéder à tous les fichiers et documents liés au projet.

2.2.1 Interface du navigateur de projet

L'interface du navigateur de projet est divisée en quatre sous-fenêtres principales, comme le montre la figure 2·8.

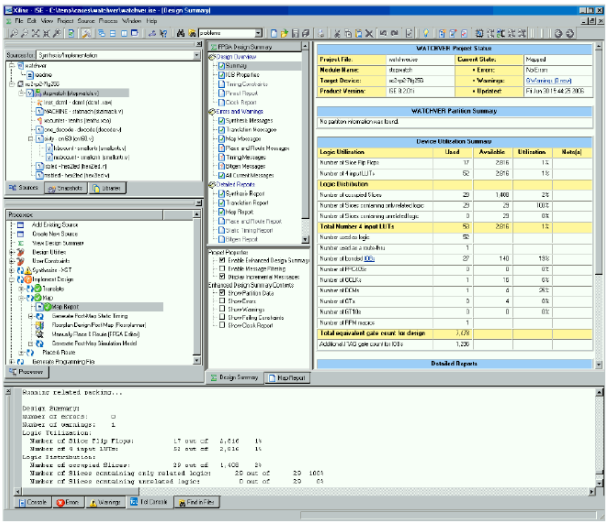

Fig. 2·8: Navigateur de projet

Sur la gauche en haut on a la fenêtre de sources qui affiche hiérarchiquement les éléments inclus dans le projet. Sous la fenêtre de sources on a la fenêtre de processus, qui affiche des processus disponibles pour la source actuellement choisie. La troisième fenêtre au bas du navigateur de projet est la fenêtre de transcription qui affiche les messages d'états, d'erreurs, et d'avertissements et contient également un éditeur de commande TCL et la fonction de recherche de fichier. La quatrième fenêtre vers la droite est l'interface d'une fenêtre pour multi document (MDI) référencé comme zone de travail. Elle permet de visualiser des fichiers d'états de type $HTML$, de textes$ASCII$, des schémas, et les formes d'onde de la simulation .

2.2.2 Conception du projet à l'aide du VHDL

Lancer le logiciel ISE

Pour lancer ISE :

Double clic l'icône du navigateur de projet d'ISE sur le bureau ou en faisant $Démarrer > tous\ les\ programmes > XilinxISE8.2ì > navigateur\ de\ projet.$

1. Créer un nouveau projet
2. À partir du navigateur de projet, choisir $file > new\ project$. La nouvelle fenêtre de projet apparaît.
3. Entrez le nom du projet.
4. Choisir un répertoire pour le projet.
5. Vérifient que HDL est choisi comme module de niveau supérieurs et cliquer suivant. La figure 2·9 est une Fenêtre de création de projet

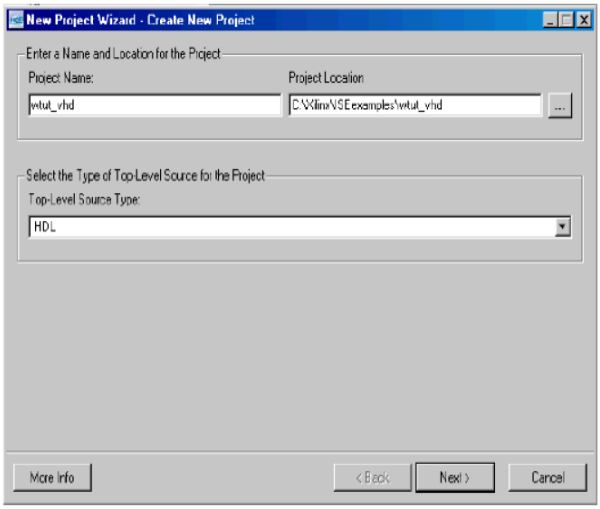

Fig. 2·9: Fenêtre de création de projet

6. Cliquer suivant afin de procéder à la création d'une nouvelle source pour le projet.
7. Choisir des valeurs pour le dispositif dans la fenêtre de propriétés de dispositif

Product Category	Tous
Family	$Virtex2$
Device	$XC2V1000$
Package	$FG456$
Speed	-4
outil de Synthesis	XST(VHDL/Verilog)
Simulator	$Modelsim - SE\ Mixed$

Notation 1

Pour l'outil de synthèse on peut si disponible choisir d'autres outils de synthèse qui ne sont pas inclut dans l'ISE mais qui sont prise en charge par l'ISE comme Spectrum synthesis tool, ces outils de synthèse sont à acheter en dehors de l'ISE. De même pour le simulateur on peut utiliser les simulateurs de mentor graphics qui sont aussi prises en charge par l'ISE et d'ailleurs des versions intérieures de l'ISE ne disposent pas de leur propre simulateur.

Créer une source HDL

Le dossier supérieur de la conception doit être en HDL. Créer une source $VHDL$

Créer un fichier source de $VHDL$ pour le projet comme suit :

1. Cliquer sur *new source* dans la fenêtre (figure 2·10).

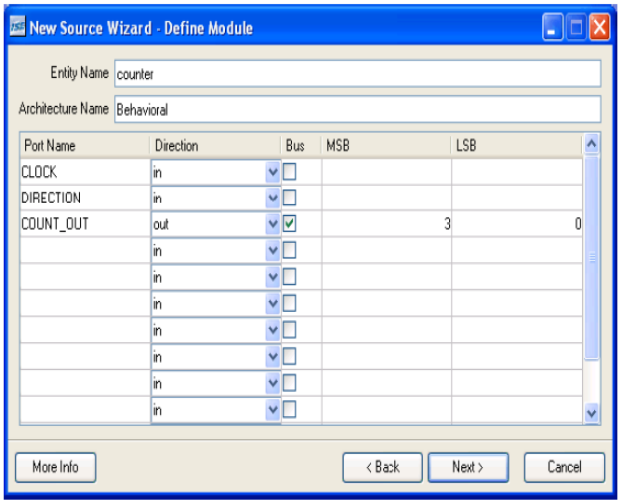

Fig. 2·10: l'insertion des ports

2. Choisir $Module\ VHDL$ comme type de source.
3. Saisir le nom du fichier.
4. vérifient que *Add to Project* est sélectionné.
5. Cliquer suivant.
6. Déclarer les ports pour la conception en complétant l'information si nécessaire le nombre de bit, MSB (most significant bit) correspond au bit de poids élevé et LSB (least significant bit) au bit de poids faible.
7. Cliquer sur $Next$ puis $Finish$ pour quitter la fenêtre d'information sur la source.
8. Cliquer sur $Next$, $Next$ puis $Finish$ pour finir.

On aura le fichier source à compléter.

Employer des références de Langue (*VHDL*)

Comme dans la plupart des conceptions on utilise généralement une combinaison d'un certains nombre de modules telles que les compteurs synchrones ou asynchrones, les registres, les flips flops, les *ROMs* et *RAMs*... C'est dans se sens que *Xilinx* a développé certaines boîtes qui nous permettent d'utiliser ces modules comme étant déjà prédéfinis et qu'on aura a adapté à notre projet. Ces modules ont été prédéfinis pour chaque type de langage et pour le type de source. Ainsi on n'a pas besoin de construire des sources pour ces modules, on peut les prendre directement dans l'*ISE* en utilisant la commande *Template language* (exemple sur la figure 2·11).

La prochaine étape explique comment utilisé ces sources
On peut remarquer que les ports sont nommés par défaut *HEX* et *LED* .

1. Placer le curseur juste a l'endroit ou l'on veut ajouter le code (généralement en dessous du *Begin* d'*architecture* pour la synthèse).
2. Ouvrir le gabarit des langues en utilisant *Edit* > *Language Templates*...

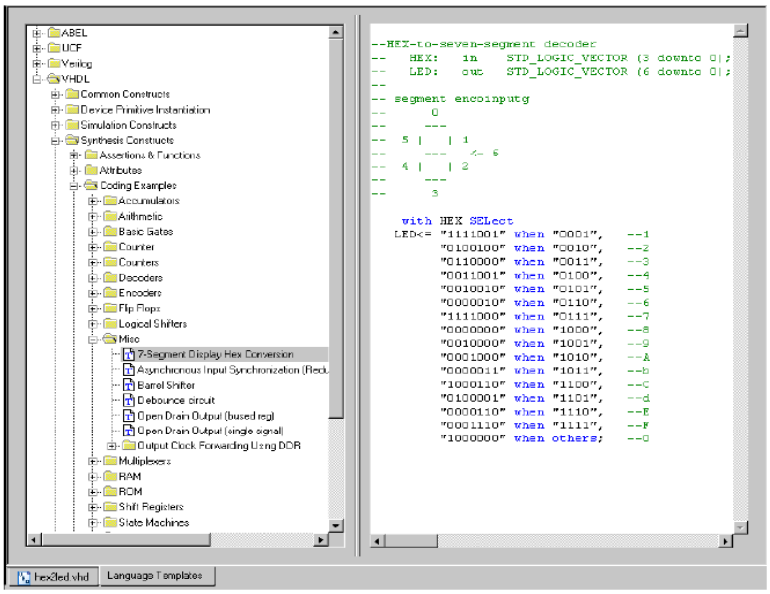

Fig. 2·11: L'example d'un codeur Hexadécimal / 7 segments de l'ISE

3. Employer cliquer sur le «+» du langage choisit, et choisir la catégorie de notre code c.-à-d. entité, architecture, attributs...
4. La source sélectionner s'ouvrira et on peut voir le code. Une fois la source sélectionnée faire *Edit* > *Use in File*, Le code s'affichera directement en dessous de l'endroit ou était placé le curseur.
5. Fermer la fenêtre et revenir au fichier dans le quelle le code a été copié pour l'adapter au reste (Généralement c'est de changé les noms des entrées et des sorties utilisées par défaut par ceux de notre projet).

Synthèse à l'aide du XST (Xilinx Synthesis Technologie)

Pendant la synthèse, les fichiers de HDL sont traduits en portes et optimalisés à l'architecture de cible. Les processus disponibles pour la synthèse utilisant XST sont comme suit :

- Rapport sur l'état de la synthèse : Donne un sommaire sur le plan et la synchronisation de la synthèse aussi bien que les optimisations qui ont eu lieu.
- Schéma de la Vue RTL : Génère une vue schématique.
- Schéma de vue de la Technologie : Génère une vue schématique de la technologie.
- Contrôler la Syntaxe : Vérifie que la source HDL est écrite correctement.

XST supporte une syntaxe de modèle du type fichier de contrainte utilisateur (UCF = User Constraints File) pour définir des contraintes de synthèse et de synchronisation. Ce format s'appelle le fichier de contrainte de $Xilinx$ (XCF = Xilinx Constraints File), et le fichier a une extension de type xcf. XST emploie l'extension xcf pour déterminer si le fichier est un fichier de contraintes. On crée les fichiers de contraintes comme pour les modules, mais il faut savoir qu'il y'a plusieurs types de contraintes et plusieurs façons de les générés.

2.2.3 Conception du projet à l'aide du schématique

Pour commencer un nouveau projet avec l'ISE pour le schématique on utilise la même procédure vue en à la seule différence il faut prendre pour module de base schématique au lieu de HDL (voir figure 2·12) dans la fenêtre de création de projet.

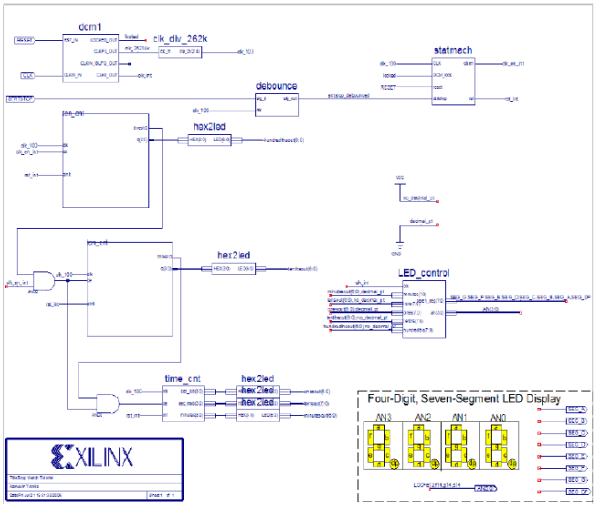

Fig. 2·12: Une conception shématique

Description de la Conception

La conception basée sur le schématique comme pour les HDL utilise un fichier supérieur de conception qui est une feuille schématique qui peut se rapporter à plusieurs autres macros instructions plus élémentaires. Les macros instructions plus élémentaires sont une variété de différents types de modules, incluant des modules schématiques, un module du générateur de $NOYAU$, et des modules HDL. Ici nous allons nous intéressé seulement aux modules schématique, puisque le générateur de noyau et les HDL ont été déjà vue dans la conception à l'aide de HDL.

Entrée de la Conception

Les entrées de la conception correspondent aux différents modules que l'on doit insérer dans une conception, se sont des modules schématiques, HDL, des noyaux, et/ou des machines d'états. Ici nous allons présenter le module schématique. Créer une macro instruction basée sur le schématique. Une macro instruction à l'aide de schématique se compose d'un symbole et d'un sous fichier schématique. On peut créer le schéma fondamental ou le symbole d'abord en premier. Le symbole correspondant ou le fichier schématique peut alors être générer automatiquement.

Dans les étapes suivantes, on crée une macro instruction basé sur le schématique en utilisant la boîte de création de source du navigateur de projet. Un fichier schématique vide est alors créé, et on peut définir la logique appropriée. La macro instruction créé alors est automatiquement ajoutée à la bibliothèque du projet. Pour créer une macro instruction

basé sur le schématique :

1. Dans le navigateur de projet, choisir $Project > newsource$. La boite de dialogue de source s'ouvre : La boite de dialogue de source affiche une liste de tous les types disponibles de source.
2. Choisir *schematic* comme type de source.
3. Ecrire le nom du fichier.
4. Cliquer sur $Next$ puis $Finish$.

Un nouveau fichier schématique est créé, ajouté au projet, et ouvert pour être éditer. Définir le fichier schématique Maintenant le fichier est créé, mais il est vide. La prochaine étape est d'ajouter les composants que doit contenir la source. Définir les ports d'entrées/sorties.

Les marqueurs d'E/S sont employés pour déterminer les ports sur une macro instruction. Le nom de chaque broche sur le symbole doit avoir un connecteur correspondant dans le sous fichier schématique. Ajouter les marqueurs d'E/S au schéma pour déterminer les macros ports. Pour ajouter les marqueurs d'E/S :

1. Sélectionner $Tools > Create\ I/O\ Markers$. La boite de dialogue pour créer des marqueurs E/S s'ouvre (Figure 2·13).

2. Dans la zone *Inputs* indiquer les noms des ports d'entrées en les séparant par des virgules.
3. Dans la zone *Outputs* indiquer les noms des ports de sorties en les séparant par des virgules.
4. Cliquer *Ok* et les ports apparaitrons sur le schéma.

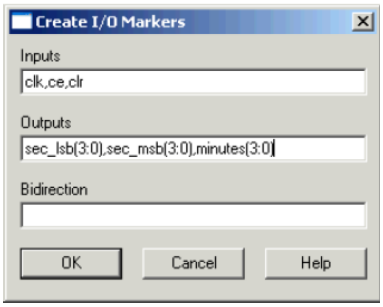

Fig. 2·13: Marqueurs d'entrées/sorties

Ajouter Des Bus

Dans l'éditeur schématique, un bus est simplement un fils multi-bit. Pour ajouter un bus, on utilise même la méthode que pour ajouter des fils et puis ajoutent un nom de multi-bit. Une fois que le bus a été créé, on a l'option "tapping" pour utiliser chaque signal du bus individuellement.

Ajouter des prises de Bus

Employer des fils pour se connecter à un seul bit d'un bus.
Pour lier un fils à un bit simple du bus :

1. choisi *add* > *bus tap* ou cliqué l'icône d'*ajout de prise de bus* dans la barre d'outils. Le curseur change, indiquant que tu es maintenant dans la mode de prise de bu

2. Avec l'option tab à la gauche du schéma, choisir l'orientation correcte pour la prise de bus.
3. Placé la prise sur un des trois bus de sorte que le côté de fil de la prise de bus se dirige vers une broche non liée.
4. répètent l'étape à tous les bits nécessaires du bus.

2.3 Simulation à l'aide du ModelSim

Pour être efficace, une simulation doit être la plus complète possible, sans être redondante. Une bonne simulation épurée de tous les tests doublés permettra une bonne économie tant pour le temps consacré à cette simulation que pour les tests sur circuit ensuite. *Vue d'ensemble de la simulation comportemental*

L'*ISE* de *Xilinx* permet pour une simulation d'intégré le simulateur *ModelSim* de mentor et le simulateur de *Xilinx ISE* qui permet à des simulations d'être exécuter sur le navigateur de projet de *Xilinx*.

ISE intègre pleinement le simulateur de *ModelSim*. *ISE* permet à *ModelSim* de créer l'espace de travail, compile les fichiers source, charger la conception, et effectuer la simulation basée sur des propriétés de simulation.

Pour choisir *ModelSim* en tant que votre simulateur de projet :

1. Dans l'étiquette de sources, clic droit sur la ligne du dispositif.
2. Choisir *Proprieties*.
3. Dans le domaine de simulateur de la zone de dialogue de propriétés de projet, choisir le type de *ModelSim* avec le langage *HDL* utilisé.

Les procédés de simulation dans *ISE* permettent d'exécuter la simulation sur la conception en utilisant *ModelSim*. Pour localiser les processus du simulateur de *ModelSim* (Figure 2·14) :

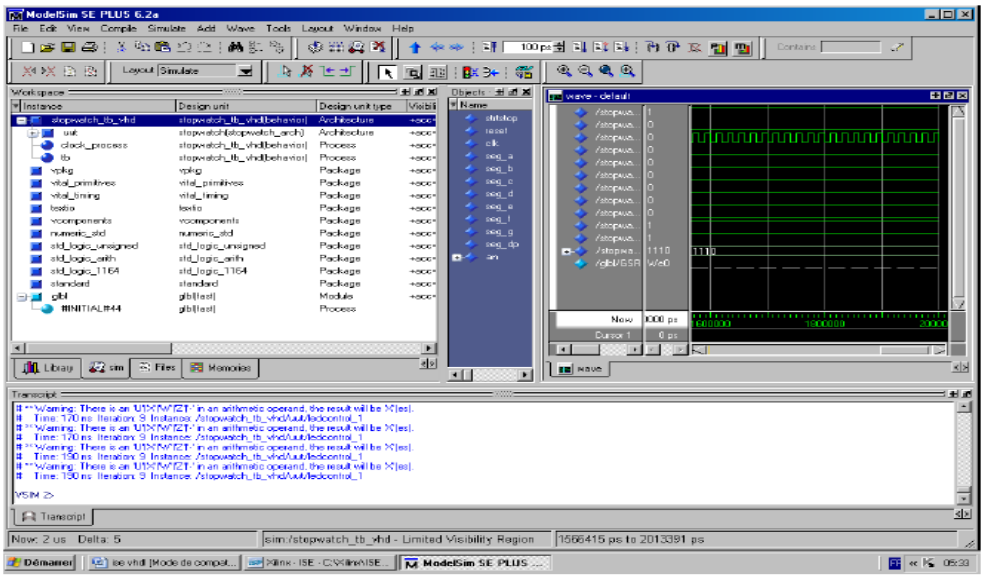

Fig. 2·14: L'environnement du ModelSim 6.2a

1. Dans la table des sources, choisir *simulation comportementale* pour le champ de la vue.
2. choisir le fichier *test bench*.
3. Dans la fenêtre de processus, cliquer sur + près du *simulateur du ModelSim* pour augmenter la hiérarchie de processus.

2.4 Implémentation

L'exécution de l'implémentation est le processus de la traduction, dresser la carte, pour le placement, routage, et la génération du fichier binaire pour la conception. Les outils d'implémentation de la conception sont dans le logiciel d'*ISE* pour faciliter l'accès et la gestion du projet (Figure 2·15).

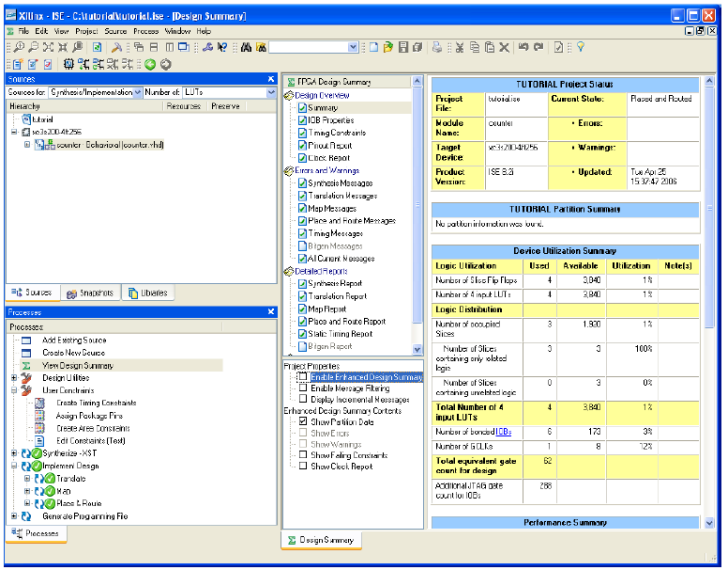

Fig. 2·15: L'environnement ISE avec les signes au niveau des processus d'Implémentation

Le placement est la phase de mise en place des différents blocs logiques obtenus par synthèse sur le dispositif.

Le routage est la phase de connexion de ses blocs sur le circuit.

1. Choisir le fichier source dans la fenêtre de sources.
2. Ouvrir le sommaire de la conception en cliquant 2 fois sur *View Design Summary* dans la fenêtre de processus.
3. Double clic *Implement Design* dans la fenêtre des processus.

2.4.1 Assigner les contraintes de location de pin

A ce stade il faut bien connaitre le dispositif pour bien mettre les *pins* (Figure 2·16). Indiquer les endroits pour les ports de la conception de sorte qu'elles soient reliées correctement à la carte.

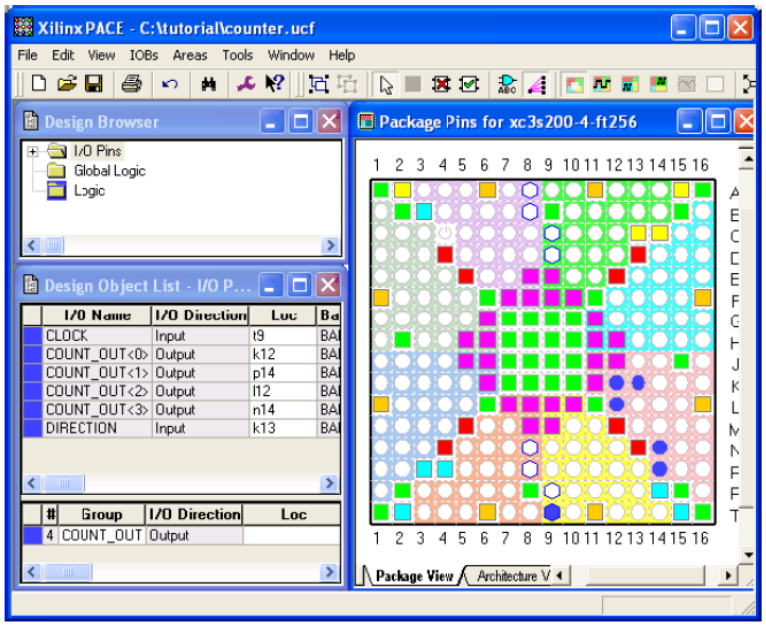

Fig. 2·16: Vue de la carte

Pour contraindre les ports de conception pour empaqueter des *pins*, on fait :

1. Vérifier que la source de niveau haut est choisie dans la fenêtre de sources.
2. Double clic sur *Assign Package Pins* le processus se trouve dans le groupe de processus de contraintes utilisateur. Le *Xilinx Pinout and Area Constraints Editor* (*PACE*) s'ouvre.
3. Choisir la table de vue de paquet *Package View*.
4. Dans la fenêtre de liste d'objet de conception, entrer le pin correspondant à chaque port au niveau de *Loc*.
5. On enregistre et on ferme la fenêtre.
6. Ré-implémenter la conception

2.4.2 Téléchargement de la conception sur la carte

C'est la dernière étape dans la procédure de vérification de la conception

1. Connecter le câble d'alimentation de $5V$ DC à la carte (j4).
2. Connecter le câble de téléchargement entre le PC et la carte (j7).
3. Sélectionner Synthesis/implementation dans la fenêtre de source.
4. Sélectionner le Module de haut niveau dans la fenêtre de source.
5. Dans la fenêtre de processus cliquer sur + à coté de *Generate Programming File*.
6. Double clic sur *Configure Device* (*iMPACT*) (Figure 2·17).

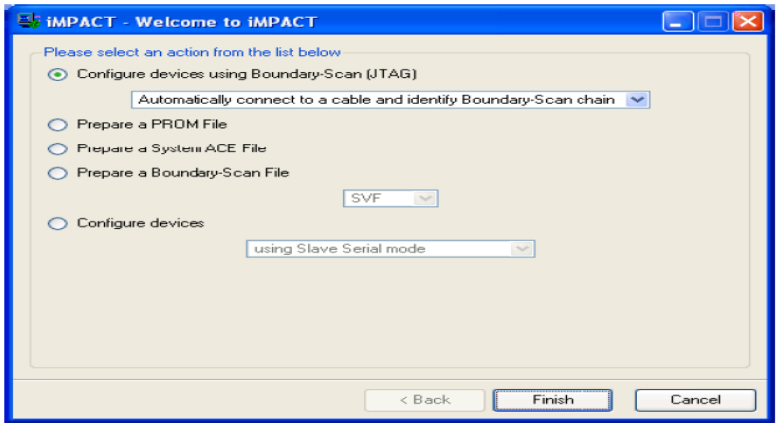

Fig. 2·17: La boite de dialogue d'IMPACT

7. La boîte de dialogue de *Xilinx* WebTalk s'ouvrira, cliquer sur *Decline*.
8. Sélectionner *Disable the collection of device usage statistics for this project only* et cliquer *OK*. IMPACT s'ouvre et on a la configuration du dispositif dans la boîte de dialogue (Figure 2·18).
9. Dan la boite de dialogue de bienvenue, sélectionner *Configure devices using Boundary-Scan* (*JTAG*).
10. Vérifier que *Automatically connect to a cable and identify Boundary-Scan chain* est selectionner.
11. Cliquer sur *Finish*.
12. S'il y'a un message qui dit que 2 dispositifs ont été trouvés cliquer *OK*.
13. La boite de dialogue *Assign New Configuration File* apparait. Pour donner le fichier de la configuration du dispositif dans la chaine du JTAG. Choisir le nom_du_module.bit et cliquer sur *OPEN*.
14. S'il y'a un message d'avertissement cliquer *OK*.
15. Sélectionner *Bypass* pour quitter les autres dispositifs.

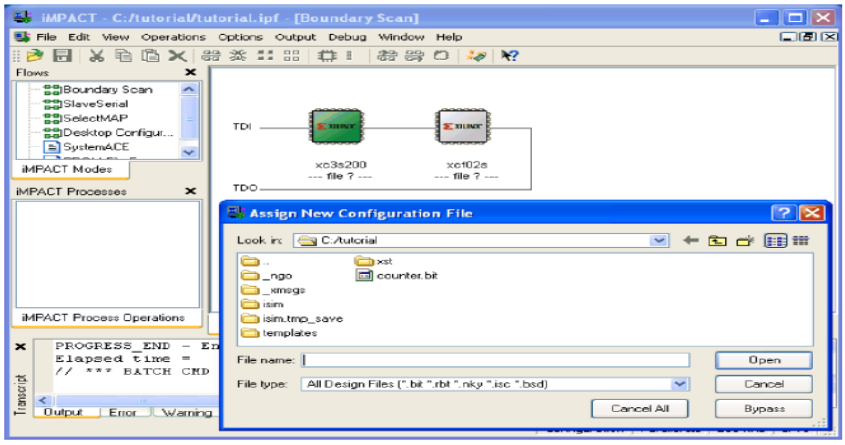

Fig. 2·18: Téléchargement du fichier binaire (nom_du_module.bit)

16. Clic droit sur l'image du dispositif, et sélectionner *Program...* la boite de dialogue du *Programming Properties* s'ouvre.
17. Cliquer *OK* dans le programme du dispositif.
18. Quand le programme sera complètement terminer, et que ça à marcher un message s'affiche.
19. Fermer *IMPACT* sans sauvegarder.

2.5 Conclusion

En premier point, nous avons vue que le monde de la communication sans fils s'intéresse très particulièrement au *FPGA*, ceci est du a leur flexibilité en utilisation. Les trois travaux présentés ici on été basé sur différentes normes, mais au coeur de tout ceci on retrouve l'*OFDM* qui est la technique presque adopté par la nouvelle génération des sans fils.

Nous avons donné aussi un aperçu sur l'utilisation de l'environnement de travail de l'*ISE*, sans pour autant trop nous approfondir car c'est logiciel très complexe et qui contient énormément d'outils d'aide à la conception. La raison de sa complexité est de vouloir répondre à toutes les étapes nécessaires à la conception, et le plus important c'est qu'il se suffit à lui-même, donc avec le logiciel on peut aborder n'importe quelle conception sur les FPGAs. Certaines parties n'ont pas été abordées comme la création de fichier de contraintes du fait que l'aborder nous emmènerais sur un chemin très long et compliqué à saisir sans certaines connaissance préalable sur le dispositif.

Chapitre 3

Architecture des blocs d'un Système OFDM

Notre implémentation du standards *WiMaxIEEE802.16d* qui exploite la technologie de partage de ressources radio *OFDM* va se faire sur trois étapes :

- Emetteur ;
- Canal ;
- Et le récepteur (voir Figure **3.1**)

Basiquement, un système *OFDM* comporte des convertisseurs série/parallèle et parallèle/série, un schéma du mapping/demapping du signal, a Fast Fourier Transform *FFT* et *FFTInverse*, un bloc ou l'ajout de l'intervalle de garde et un autre pour sa suppression. Le convertisseur série/parallèle a pour rôle de réaliser une transmission de donnée en parallèle pour fournir un haut débit de transmission. Le schéma du mapping de *M-arrayPSK* ou de la modulation en quadrature de phase *QAM* est modulé à chaque sous-porteuse pour fournir des débits différents dans le système. La *FFT* et l'*IFFT*sont utilisés pour remplacer les générateurs de sinusoïdes lors de la modulation et la démodulation avec des fréquences de porteuses différentes pour réduire la complexité du modem *OFDM* à implémenter.

Pour notre cas, c'est-à-dire la norme *WiMAXIEEE802.16d*, nous allons implémenter sur une carte *VIRTEX-II* le système montré comme suit :

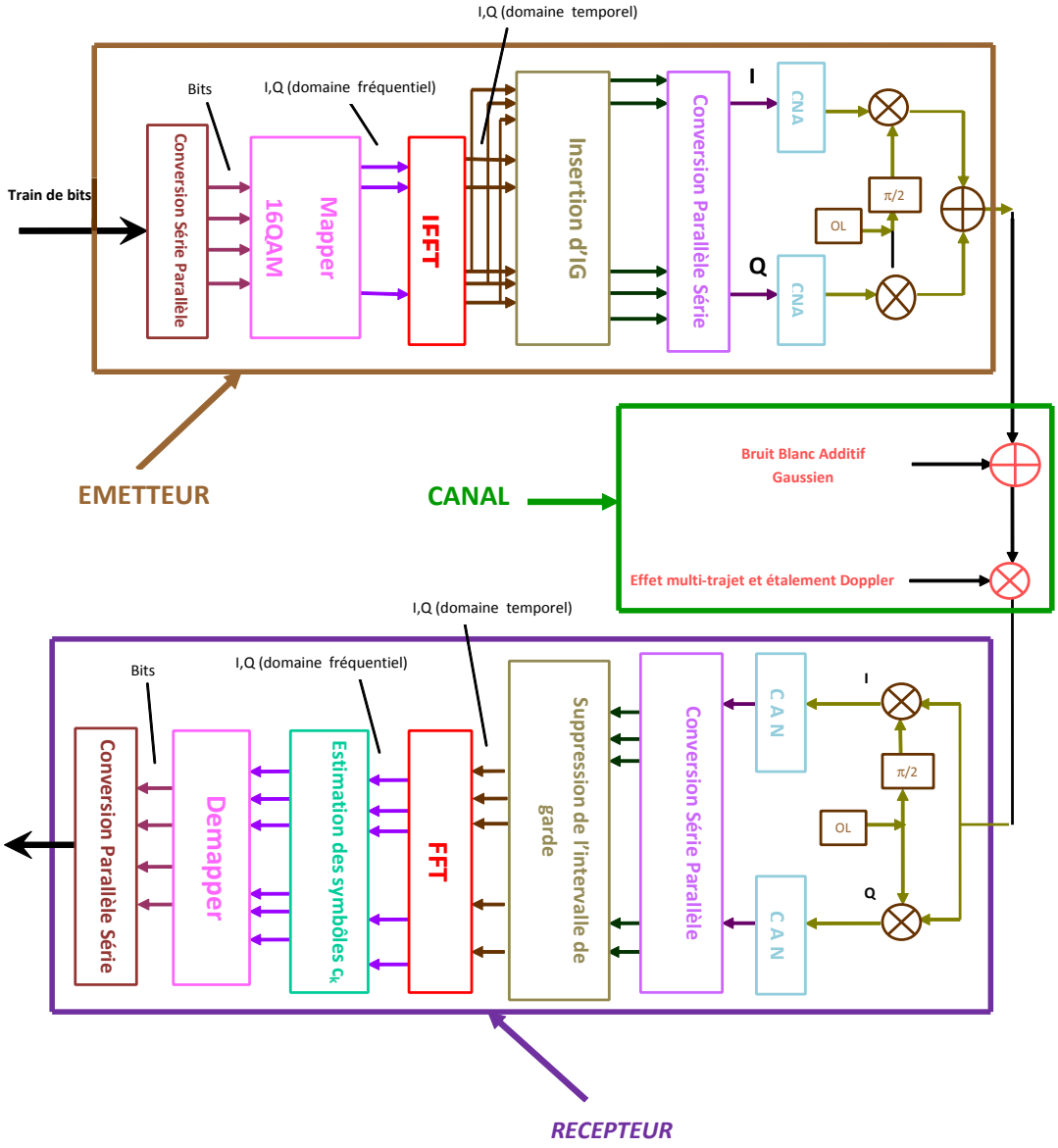

Fig. 3·1: Le système à implémenter

3.1 Emetteur :

3.1.1 Convertisseur Série/Parallèle

Une constellation de *16QAM* a pour entrée 4 bits et donc on doit utiliser un convertisseur série/parallèle 1 vers 4 (voir Figure **3.2**).

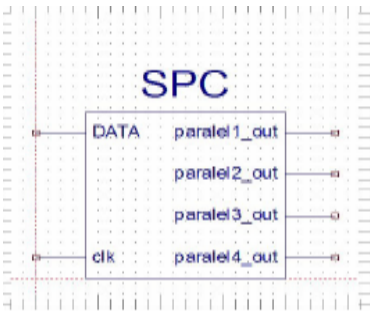

Fig. 3·2: Entité du convertisseur Série Parallèle faite par ISE 8.2i

Les quatre bascules du premier étage ont la fonction de décalage (voir Figure **3.3**).

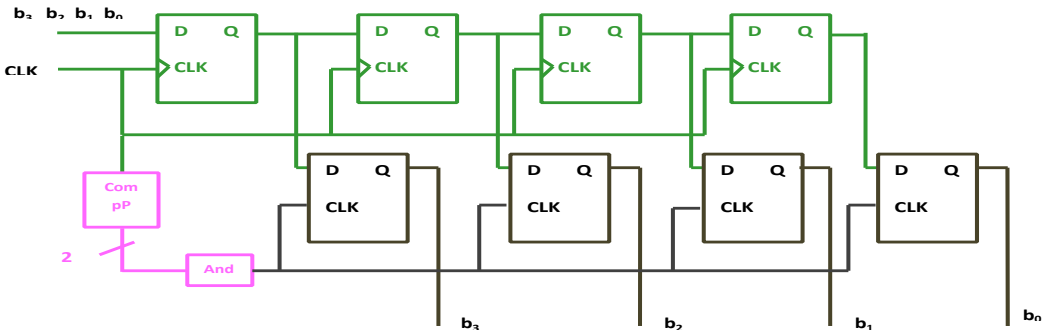

Fig. 3·3: Schéma Bloc d'un Convertisseur Série Parallèle

Après que ces 4 bascules ou registres soient chargées c'est-à-dire après le 4ème cycle d'horloge, le compteur *CompP* aurait terminé sa boucle (4) ce qui va générer un signal haut à la sortie de la porte And qui est l'horloge des 4 bascules du 2ème étage et donc elles vont se charger des données des bascules du 1er étage. Nous pouvons voir l'évolution temporelle des sorties des bascules sur le chronogramme suivant (Figure **3.4**) :

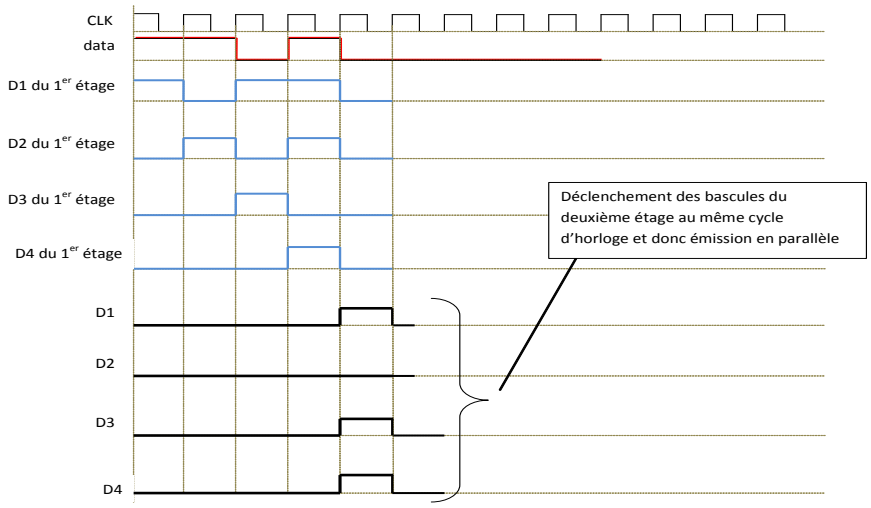

Fig. 3·4: Chronogramme du convertisseur série parallèle

3.1.2 Mapper

Pour cette implémentation nous allons se contenter de la constellation *16QAM* toujours à cause de la contrainte de la capacité de la carte. Cette représentation consiste à représenter un vecteur de 4 bit par un vecteur complexe (Figure 3·5)

Fig. 3·5: Entité du mapping fait par ISE8.2i

Les coordonnées des points de la constellation sont sauvegardées dans une *ROM* comme suit (Tableau 3.1).

Après le chargement de la *FIFO* de taille de 16 registres et 4 bits par registre, ces derniers vont définir un point de la constellation *16QAM* qui a une composante réelle I de la phase et une partie imaginaire Q de la quadrature de phase. A la sortie du mapper,

chaque ensemble de 4 bits sera représenté par un nombre complexe *(IQ)* dans le domaine fréquentiel.

$\mathbf{b_0}$	$\mathbf{b_1}$	$\mathbf{b_2}$	$\mathbf{b_3}$	SORTIES	**I**	**Q**
0	1	0	0		3	3
0	1	1	0		3	1
0	1	1	1		3	-1
0	1	0	1		3	-3
1	1	0	0		1	3
1	1	1	0		1	1
1	1	1	1		1	-1
1	1	0	1		1	-3
1	0	0	0		1	3
1	0	1	0		-1	1
1	0	1	1		-1	-1
1	0	0	1		-1	-3
0	0	0	0		-3	3
0	0	1	0		-3	1
0	0	1	1		-3	-1
0	0	0	1		-3	-3

Tab. 3.1: Les valeurs (I,Q) pour chaque vecteur de 4 bits

3.1.3 IFFT

L'IFFT 256 points (voir Figure 3·6) semble être favorable pour le standard *WiMAX* pour des raisons telles que le faible rapport de la puissance moyenne par rapport au pic_{max} (*peak to average power ratio)*, des calculs plus rapide par rapport à 64 et 128 points et des exigences moins strictes pour la fréquence de synchronisation par rapport à 2048points. L'entité Bloc IFFT comporte une entrée pour l'horloge, une entrée pour la partie réelle et une autre pour la partie imaginaire. Et comme sorties, nous avons une pour la partie réelle et une autre pour la partie imaginaire. Le schéma détaillé du bloc IFFT 256 points est représenté sur la figure 3·7.

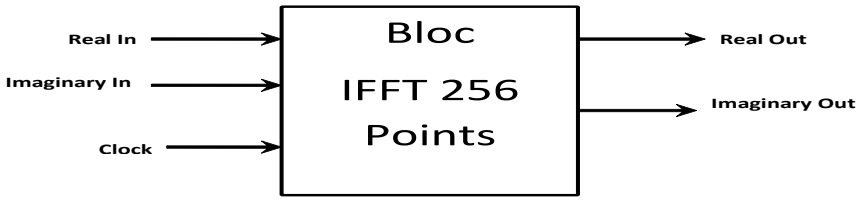

Fig. 3·6: L'entité Bloc IFFT

Les signaux de commande du processeur *IFFT*

	Positions
Pilotes	-91, -63, -35, -7, 7, 35, 63, 91
Zéros	-127, -123,-119,-115,-111, -107,-103,-99,-95,-93, -87,-83, -79,-75,-71,-67,-63,-59,-55,-51, -47,-43,-39,-35,-31,-27,-23, -19,127,123,119,115,111, 107,103,99,95,93,87,83, 79,75,71,67,63,59,55,51, 47,43,39,35,31,27,23,19.
data	

Tab. 3.2: Les positions des données, des zéros et des pilotes dans l'IFFT

Les registres internes du processeur *IFFT* se chargent après chaque cycle d'horloge par une donnée (FIFO map) ou un pilote ou un zéro. Un signal start du début des calculs se déclenchera après 256 cycles d'horloge (après la fin du chargement).

Il faut savoir aussi que les pilotes, les zéros et les données seront insérés pendant ces 256 cycles à des moments bien précis comme il était défini dans le standard *IEEE208.16d* du *WiMAX*. Sur ces 256 Sous-porteuses, 192 sont utilisés pour les données de l'utilisateur, 56 pour des zéros et 8 sont utilisés comme pilotes pour l'estimation du canal. Le positionnement des données, des zéros et des pilotes dans les registres internes du processeur IFFT est donné par Tableau 3.2

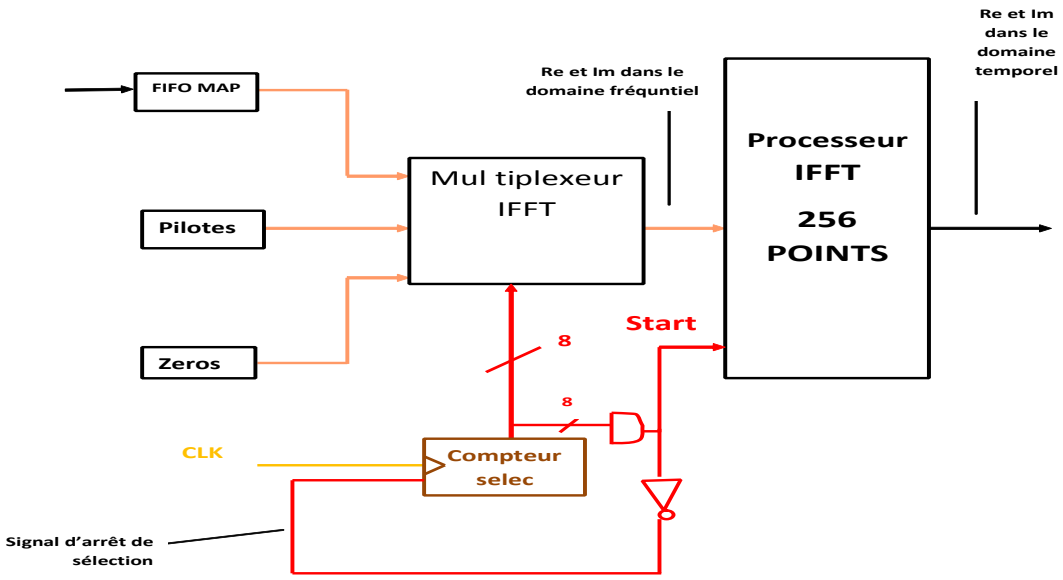

Fig. 3·7: Le schéma bloc de l'entité IFFT

	Positions
Pilotes	37,65,93,121,135,163,191,219
Zéros	1,5,9,13,17,21,25,29,33,37,41, 45,49,53,57,61,65,69, 73,77,81,85,89,93,97,101, 105,109,113,147,151,155,159, 163,167,171,175,179,183,187, 791,195,199,203,207,211, 215,219,223, 227, 231, 235, 239, 243, 247, 251,255
data	

Tab. 3.3: Les positions des données, des zéros et des pilotes dans comIFFT

Pour sélectionner donnée, zéro ou pilote, nous allons insérer un multiplexeur qui aura comme signal de commande la sortie d'un compteur asynchrone modulo 256 « *compteurselec* » (voir Figure 3·7). Le positionnement des données, zéros et pilotes dans le compteur compteur selec est donné par le tableau 3.3

Une fois que le comptage ait atteint 256 cycles, un signal de validation du commencement des calculs sera envoyé et le multiplexeur sera forcé d'arrêter la sélection. Les données, avant qu'elles ne soient envoyées à l'*IFFT* à travers le multiplexeur, elles seront stockées dans une *RAM* (FIFO map) pour ne pas freiner le processus du mapping. A la sortie du processeur *IFFT*, un vecteur de 256 éléments est généré où chaque élément est l'image complexe d'un symbole d'information.

Traitement du processeur IFFT

Le nombre de bits disponible dans le processeur de l'*IFFT* est 8 bits. La longueur d'un mot de 8 bits représente 256 valeurs ($2^8 = 256$).(**2**,)

Les additionneurs binaires et multiplicateurs binaires sont les composantes principales qui constituent le processeur *IFFT*.(**1**,)

L'expression générale de la transformée de fourier rapide est (equation 3.1) :

$$y[k] = \sum_{k=0}^{N-1} Y[n].W_N^{nk} \tag{3.1}$$

où W_N (Twiddle Factor) est égale à $e^{\frac{2\pi j}{N}}$

En séparant les échantillons Y[n] pairs et impairs, l'équation 3.1 devient :

$$y[k] = \sum_{k=0}^{\frac{N}{2}-1} Y[2n]W_N^{2nk} + \sum_{k=0}^{N-1} Y[2n+1]W_N^{(2n+1)k} \tag{3.2}$$

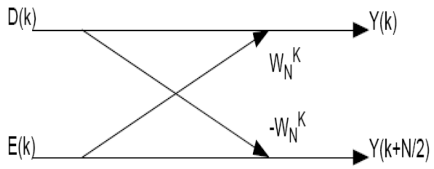

Fig. 3·8: Unité BUTTERFLY radix 2

On met en facteurW_N dans le terme des échantillons impairs, l'équation 3.2 devient :

$$y[k] = \sum_{k=0}^{\frac{N}{2}-1} Y[2n] W_N^{2nk} + W_N^k \left(\sum_{k=0}^{N-1} Y[2n+1] W_N^{2nk} \right) \tag{3.3}$$

Le développement du terme W_N^{2nk} donne :

$$W_N^{2nk} = \left(e^{nk\left(-j\frac{2\pi}{N}\right)} \right)^2 = e^{nk\left(-j\frac{2\pi}{N}\right).2} = e^{nk\left(-j\frac{2\pi}{N/2}\right)} = W_{N/2}^{nk} \tag{3.4}$$

L'équation 3.3 peut s'écrire alors :

$$y[k] = \sum_{k=0}^{\frac{N}{2}-1} Y[2n] W_{N/2}^{nk} + W_N^k \left(\sum_{k=0}^{N-1} Y[2n+1] W_{N/2}^{nk} \right) \tag{3.5}$$

y(k) devient :

$$y[k] = D[k] + W_N^k E[k] \tag{3.6}$$

où $D[k] = \sum_{k=0}^{\frac{N}{2}-1} Y[2n] W_{N/2}^{nk}$ et $E[k] = \left(\sum_{k=0}^{N-1} Y[2n+1] W_{N/2}^{nk} \right)$ tel que$k = 0, ., N-1$.

Le nombre des calculs pour l'IFFT est : $\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 = \frac{N^2}{2}$ tandis qu'en $IDFT$, le nombre de calculs est N^2.

Nous avons $D[k+\frac{N}{2}] = D[k]$ et $E[k+\frac{N}{2}] = E[k]$ et $W_N^{k+\frac{N}{2}} = -W_N^k$, par conséquent la différence entre deux membres paire symétrique ou deux membres impaires symétrique est W_N^k qui est positif pour le premier membre paire ou impaire et négatif pour le deuxième membre paire ou impaire. Ceci est représenté par une unité de calcul appelée $BUTTERFLY$ ou Papillon (voir Figure 3·8).(**1**,)

Pour illustrer cette théorie, nous avons travaillé avec la cas particulier simple N=4.

$$y[0] = D[0] + W_4^0 E[0] = D[0] + E[0] = Y[0] + Y[2] + Y[1] + Y[3] \tag{3.7}$$

$$y[1] = D[1] + W_4^1 E[1] = D[1] + W_4^1 E[1] = Y[0] - Y[2] + j\,(Y[1] - Y[3]) \tag{3.8}$$

	Réel	Imaginaire
y[0]	$Y[0]_{réel}+Y[2]_{réel}+Y[1]_{réel}+Y[3]_{réel}$	$Y[0]_{im}+Y[2]_{im}+Y[1]_{im}+Y[3]_{im}$
y[1]	$Y[0]_{réel}-Y[2]_{réel}+Y[1]_{im}-Y[3]_{im}$	$Y[0]_{im}-Y[2]_{im}+Y[1]_{réel}-Y[3]_{réel}$
y[2]	$Y[0]_{réel}+Y[2]_{réel}-Y[1]_{réel}-Y[3]_{réel}$	$Y[0]_{im}+Y[2]_{im}-Y[1]_{im}-Y[3]_{im}$
y[3]	$Y[0]_{réel}-Y[2]_{réel}+Y[1]_{im}-Y[3]_{im}$	$Y[0]_{im}-Y[2]_{im}+Y[1]_{réel}-Y[3]_{réel}$

Tab. 3.4: Les sorties de l'IFFT

$$y[2] = D[2] + W_4^2 E[2] = D[0] - E[0] = Y[0] + Y[2] - Y[1] + Y[3] \tag{3.9}$$

$$y[3] = D[3] + W_4^3 E[3] = D[1] - W_4^1 E[1] = Y[0] - Y[2] - j\,(Y[1] - Y[3]) \tag{3.10}$$

Le tableau 3.4 donne les sorties réelles et imaginaires des composantes temporelles.

Nous pouvons généraliser ce processus de calcul pour 256 points en utilisant 64×64 unités de calcul de $N = 4$ (voir Figure 3·9)

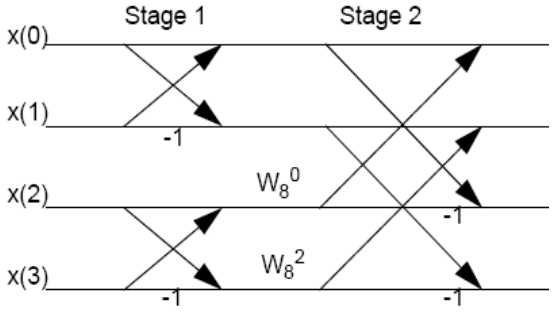

Fig. 3·9: Unité de calcul pour N=4 (comportant 4 unité Butterfly)

3.1.4 Insertion de l'Intervalle de Garde

Dans le standard *WiMAXIEEE802.16d*, on utilise un rapport de symbole de 1/16 et donc il faut insérer 16 symboles (256/16). Le bloc de l'insertion de l'intervalle de garde

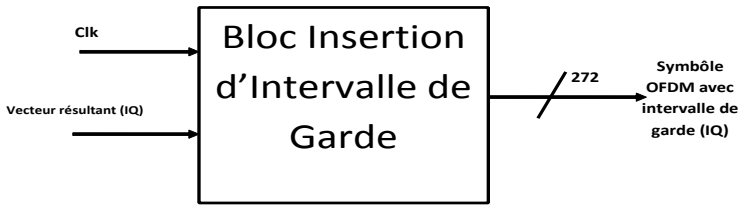

Fig. 3·10: L'entité de l'Insertion de l'Intervalle de Garde

(voir figure 3·10) est une entité qui a pour entrées un vecteur de 256 valeurs réelles et un vecteur de 256 valeurs imaginaire et une horloge. A la sortie, on a un vecteur de 272 valeurs réelles et un vecteur de 272 valeurs imaginaires.

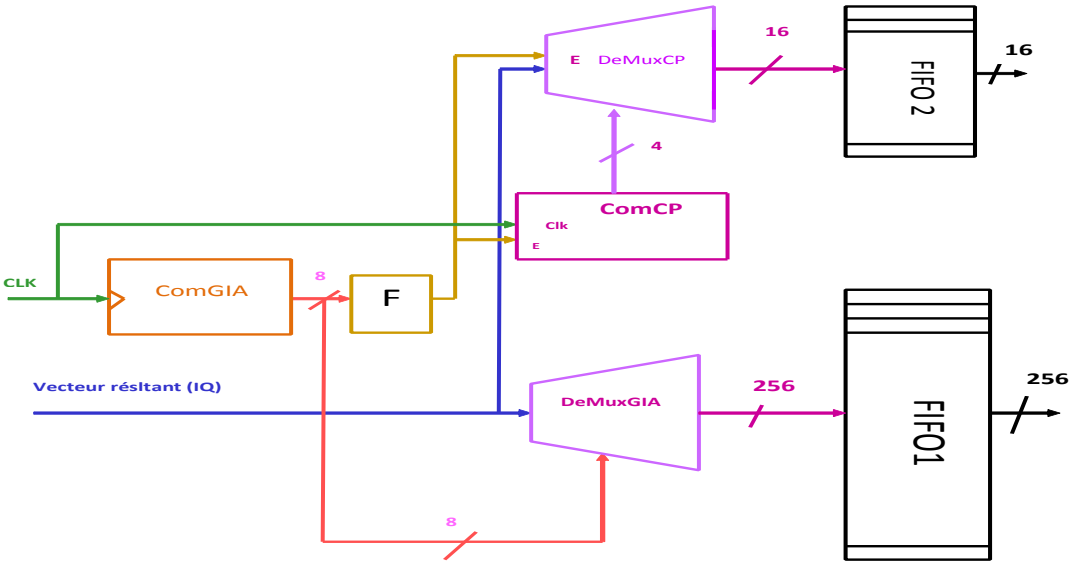

Fig. 3·11: Schéma bloc de l'insertion de l'Intervalle de Garde

Le schéma détaillé de l'entité de l'insertion de l'intervalle de garde est donné par la figure 3·11.

Les éléments du vecteur résultant de l'IFFT vont charger *FIFO1* à travers le *MuxGIA* (Figure 3·11) contrôlé par un compteur *ComGIA*. Et dès que le compteur *ComGIA* (Figure 3·11) ait atteint la position 240, il va envoyer un signal de validation au compteur *ComCP* qui contrôle le multiplexeur *MuxCp*, ce dernier va recevoir au même temps le même signal de validation du compteur à travers la fonction logique $F = b_7.b_6.b_5.b_4$.Le signal de validation E est maintenu à niveau haut pendant le chargement de la *FIFO* 2 et donc la fonction F est égale à 1 pour les combinaisons de 240 à 255.

Cette validation va entraîner le chargement des deux *FIFO* en même temps des symboles : $241, 242, 243, \ldots\ldots\ldots, 254, 255, 256$ qui représentent les éléments de l'intervalle du garde ou du préfixe cyclique. Nous utilisons deux RAM pour insérer cet intervalle, la première est de taille de 256 registres qui a pour rôle de stocker le vecteur résultant de l'IFFT et la deuxième stocke les 16 derniers symboles (voir figure 3·11) de telle sorte que l'ordre du moins significatif au plus significatif soit respecté, en d'autres termes :

- $241^{ème}$ élément du vecteur résultant sauvegardé à la $1^{ère}$ position de la FIFO 2,
- $242^{ème}$ élément soit sauvegardé à la $2^{ème}$ position de la FIFO 2,

  .....
- $256^{ème}$ élément soit sauvegardé à la $16^{ème}$ position de la FIFO 2.

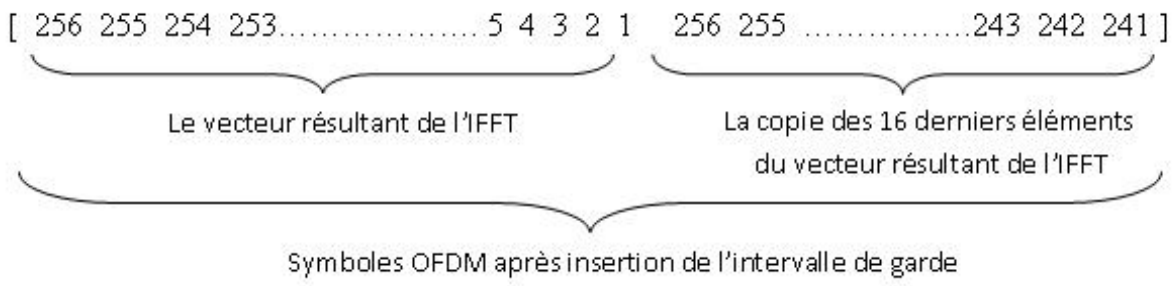

Fig. 3·12: Vecteur résultant de l'ajout de l'intervalle de garde

Les éléments de cette deuxième FIFO sont moins significatifs que ceux de la première (voir Figure 3·12)

3.1.5 Conversion Parallèle/Série

La conversion parallèle/série consiste à émettre en série ce qui était en parallèle tout en respectant l'ordre du moins significatif au plus significatif.

L'entité (voir Figure 3·13) comporte comme entrée le symbole OFDM (sortie de l'IFFT + l'intervalle de garde) qui sont en parallèle et une commande de début de conversion qui est la sortie du ComGIA de l'entité de l'insertion de l'intervalle de garde. La sortie est séquence série.

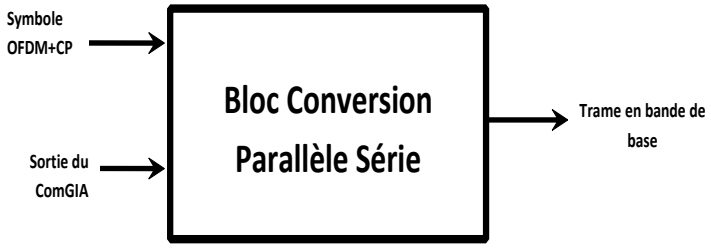

Fig. 3·13: L'entité bloc de Conversion Parallèle Série

Le schéma bloc de la conversion série parallèle est donné par la figure 3·14.

A la réception du signal de validation E qui sera à l'état haut pour la sortie du compteur ComGIA égalant $(11111111)_2$ ($b_7.b_6.b_5.b_4.b_3.b_2.b_1.b_0$=1) ou si le signal clear est à l'état bas c'est-à-dire que l'opération de sérialisation est toujours en cours (voir Figure 3·14).

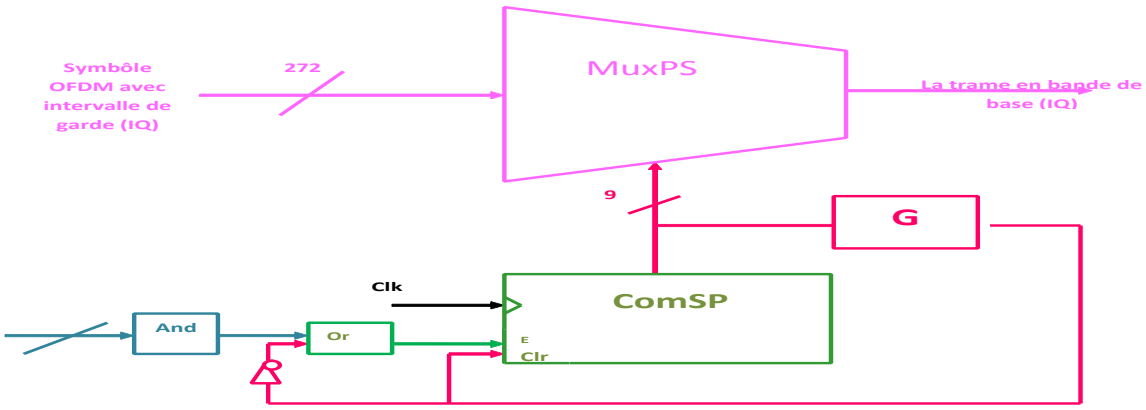

Fig. 3·14: Schéma bloc de la Conversion Parallèle Série

Le signal Clear est à l'état haut quand le compteur ComSP aurait terminé sa boucle c'est-à-dire ses 272 points (cycles d'horloge). Le signal Clear=1 est généré par la fonction logique G : G=1 pour la séquence $(271)_{10}=(100001111)_2$.

Si G=1 (on mentionne que la sortie de ComGIA sera différente de $(11111111)_2$, car elle ne dure qu'un cycle d'horloge) alors E=0 et le comptage est arrêté jusqu'à ce que l'opération prochaine d'insertion du préfixe cyclique serait terminée et donc la sortie du ComGIA=$(11111111)_2$ (voir Figure 3·14).

3.1.6 Modulation RF

Pour pouvoir émettre dans un canal radio, on module le signal OFDM avec une modulation QAM ((**20**,)). Pour le STANDARD WiMAX IEEE802.16d, la porteuse utilisée est située dans la gamme de 6 à 11GHz. Le modulateur Radio fréquence utilisé est donné par le schéma de la Figure 3·15

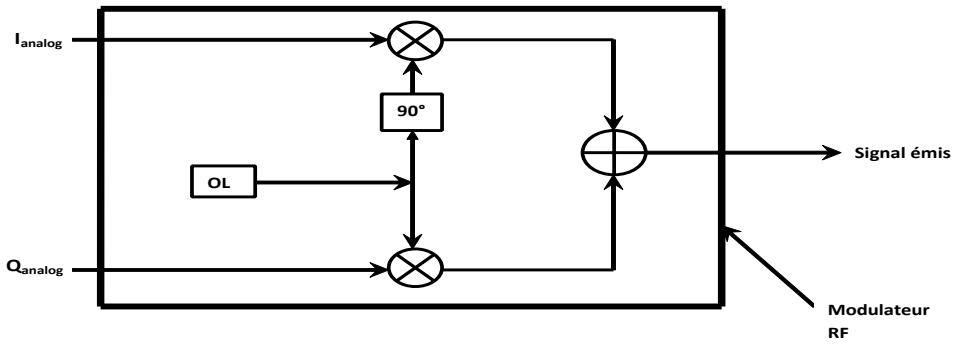

Fig. 3·15: Modulateur Radio Fréquence

Dans le code VHDL, on doit générer une sinusoïde sur un intervalle $T_s = \frac{1}{2.N}.T$, tel que N est une puissance de 2 et T la durée d'un cycle d'horloge.

Dans le cas où N=4, cette sinusoïde se répète dans un cycle d'horloge 8 fois (4 fois sur l'état haut de l'horloge et 4 fois sur l'état bas). Ceci peur être fait en utilisant la commande While ou Wait et introduisant un signal de répétition (integer range 0 to 4, clk'Event and clk=1 ensuite clk'Event and clk=0).

En VHDL, une sinusoïde en haute fréquence peut être assimilée par un signal triangulaire, car ce n'est pas évident d'implémenter un sinus ou un cosinus en Hardware. On mentionne que pour le signal modulé en cosinus (la composante imaginaire), le premier extrémum (correspondant au début du cycle T_s) doit être égal au symbole modulé. Le signal triangulaire est exprimé, pour un symbole X donné modulé en cosinus, par.

$$s(t) = \begin{cases} \begin{bmatrix} X.(1-\alpha) & 0 \prec t \preceq \frac{T_s}{4} \\ -\alpha.X & \frac{T_s}{4} \prec t \preceq \frac{T_s}{2} \\ X.(\alpha-1) & \frac{T_s}{4} \prec t \preceq \frac{T_s}{2} \\ \alpha.X & \frac{3.T_s}{4} \prec t \preceq T_s \end{bmatrix} \end{cases}$$

Où α varie de 0 à 1 pour chaque quart de cycle T_s avec un pas de 0.1.

Pour un symbole modulé en sinus, le premier extrémum correspond au premier quart du cycle T_s. Le signal triangulaire sera donné dans ce cas pour un symbole X donné modulé en sinus par :

$$\begin{bmatrix} \alpha.X & 0 \prec t \preceq \frac{T_s}{4} \\ X.(1-\alpha) & \frac{T_s}{4} \prec t \preceq \frac{T_s}{2} \\ -\alpha.X & \frac{T_s}{4} \prec t \preceq \frac{T_s}{2} \\ X.(\alpha-1) & \frac{3.T_s}{4} \prec t \preceq T_s \end{bmatrix}$$

Où α varie de 0 à 1 pour chaque quart de cycle T_s avec un pas de 0.1. On remarque que l'extrémum d'un symbole I correspond au zéro de l'autre Q.

3.2 Canal

Le canal radio mobile est exposé à de nombreuses perturbations. Pour notre implémentation, nous considérons un canal à bruit additif blanc gaussien AWGN, l'effet du multi trajet, l'étalement Doppler et les atténuations.

Pour modéliser l'effet d'un canal variant dans le temps, nous appliquons au signal émis des retards. Pour le décalage fréquentiel, nous introduirons à chaque version retardée un décalage fréquentiel. La figure 3·16 représente la modélisation d'un canal à multitrajet avec Doppler Spread et AGWN.

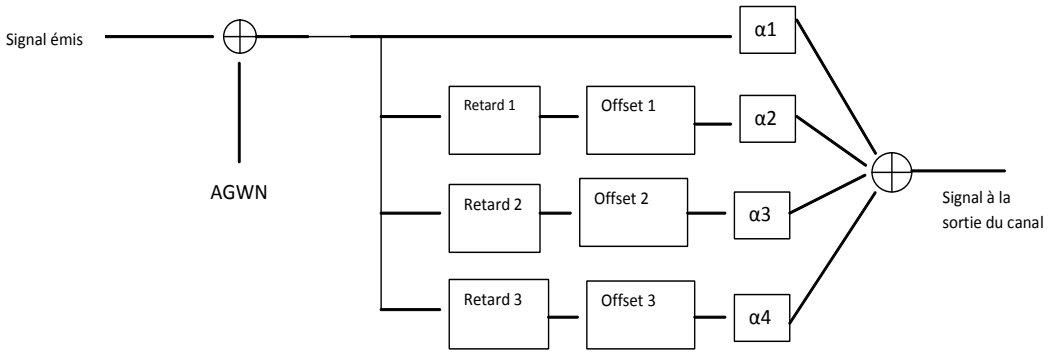

Fig. 3·16: Canal à multitrajet avec Doppler Spread et AGWN

Le schéma bloc pour la réalisation des retards est illustrer dans la figure 3·17. nous utilisons un compteur qui validera le passage du signal dans la branche i après i cycles d'horloge (Figure 3·17).

Les gains sont exprimés sous format 0.16 non signé en virgule flottante sont comme suit ***(1)***

- $\alpha_1 = (0.7)_{10} = (.1011001000000000)_2 = (.B200)_{16}$
- $\alpha_2 = (0.5)_{10} = (.1000000000000000)_2 = (.8000)_{16}$
- $\alpha_3 = (0.3)_{10} = (.0100110100111000)_2 = (.5C38)_{16}$
- $\alpha_4 = (0.2)_{10} = (.0011100100000000)_2 = (.3900)_{16}$

Le retard est exprimé comme suit $t_r = \dfrac{n}{f_{clk}}$, où n est le nombre de cycles et f_{clk} est la fréquence de l'horloge de valeur de $1Mhz$. Nous générons les trois retards suivants :

- $Retard_1 = 2microsec => n = 80$
- $Retard_2 = 3microsec => n = 120$
- $Retard_3 = 5microsec => n = 200$

A cause des contraintes matérielles, le retard doit être compris entre : $64ns \leq tr \leq 6.4\mu s$

Le signal va se mettre aux trois portes and, les compteur1, 2 et 3 vont démarrer à compter au même instant pour le signal E_Delay=1. Quand le Comp1 aurait atteint $(79)_{10}$, La fonction D_1 se met à l'état 1 (le signal subit un retard de 80 cycles d'horloge équivalent à $2\mu s$). Entre temps, les registres du compteur Comp1 se figent à la valeur $(79)_{10}$. Un signal Reset met les registres à 0 pour recommencer à retarder à nouveau le signal. On procédera de la même façon pour les autres versions retardées du signal. Les fonctions logiques D_1, D_2 et D_3 sont définies :

- $D_1 = b_0.b_1.b_2.b_3.not\,(b_4)\,.not\,(b_5)\,.b_6.not\,(b_7)$
- $D_2 = b_0.b_1.b_2.not\,(b_3)\,.b_4.b_5.b_6.not\,(b_7)$
- $D_3 = b_0.b_1.b_2.not\,(b_3)\,.not\,(b_4)\,.not\,(b_5)\,.b_6.b_7$

Pour représenter le décalage fréquentiel, On applique notre signal à l'entrée d'une boite dont la fréquence d'horloge est différente de celle avec laquelle il était émis dans à la

sortie du modulateur RF. Dans le Xilinx, il existe des diviseurs de fréquence. Pour cette implémentation, nous utilisons le même principe que celui de la modulation pour varier la fréquence du signal dans le canal..

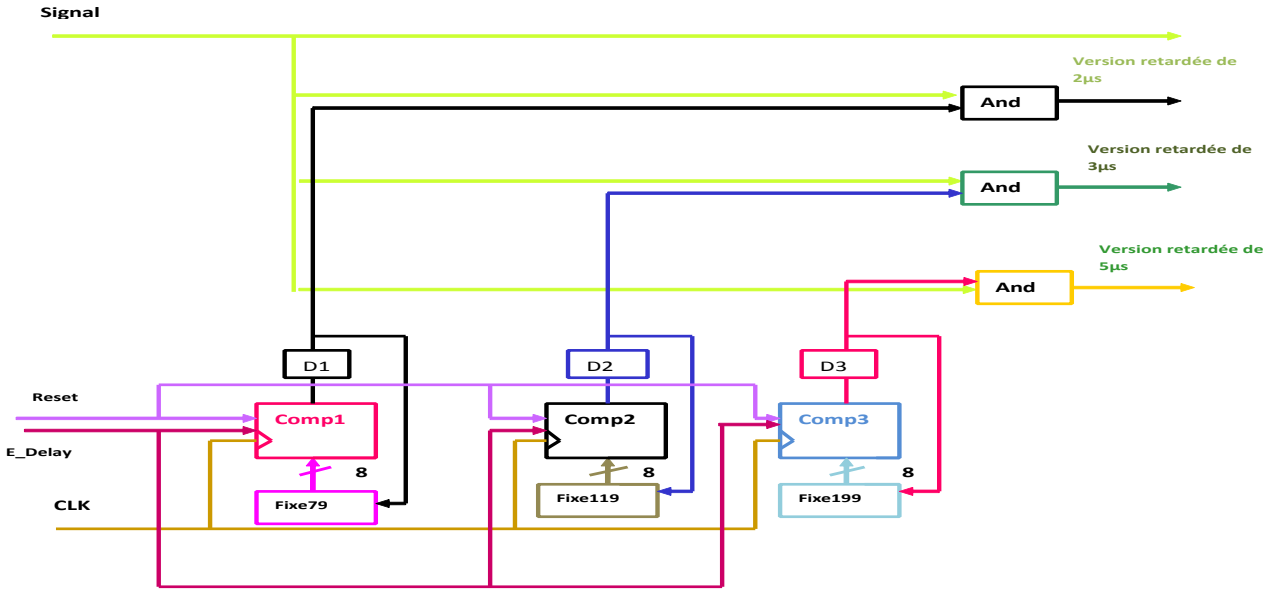

Fig. 3·17: Schéma bloc pour réaliser les versions retardées

3.3 Récepteur

En réception, on réalise les opérations inverses effectuées en émission. Le récepteur se compose des blocs suivants : FFT, Demapper, suppression de l'intervalle de garde.... On note aussi les opérations de synchronisation, l'égalisation de canal et estimation des symboles.

3.3.1 Démodulation et récupération du vecteur IQ

La démodulation consiste à récupérer les symboles OFDM en baseband, partie réelle I et partie imaginaire Q comme le montre la figure 3·18 .

Dans le cas N=4 (donc la fréquence de modulation est 8 fois la fréquence de l'horloge). Pour récupérer une composante réelle modulée sur un sinus, on prend la valeur du premier quart du cycle du signal modulé ou encore le premier 1/ $32^{ème}$ du cycle d'horloge et l'étaler sur tout le cycle (d'horloge). Pour récupérer une composante imaginaire, on prend la valeur du premier 0 du cycle du signal reçu et l'étaler sur tout le cycle d'horloge (**20**,).

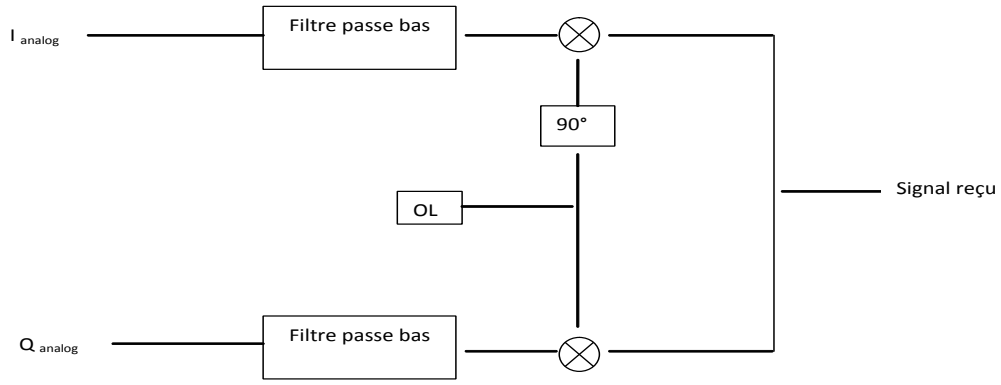

Fig. 3·18: Démodulation Radio Fréquence

3.3.2 Conversion Série Parallèle

La Figure 3·19 montre le schéma bloc à implémenter de la conversion série parallèle pour la composante I et la composante Q du signal démodulé.

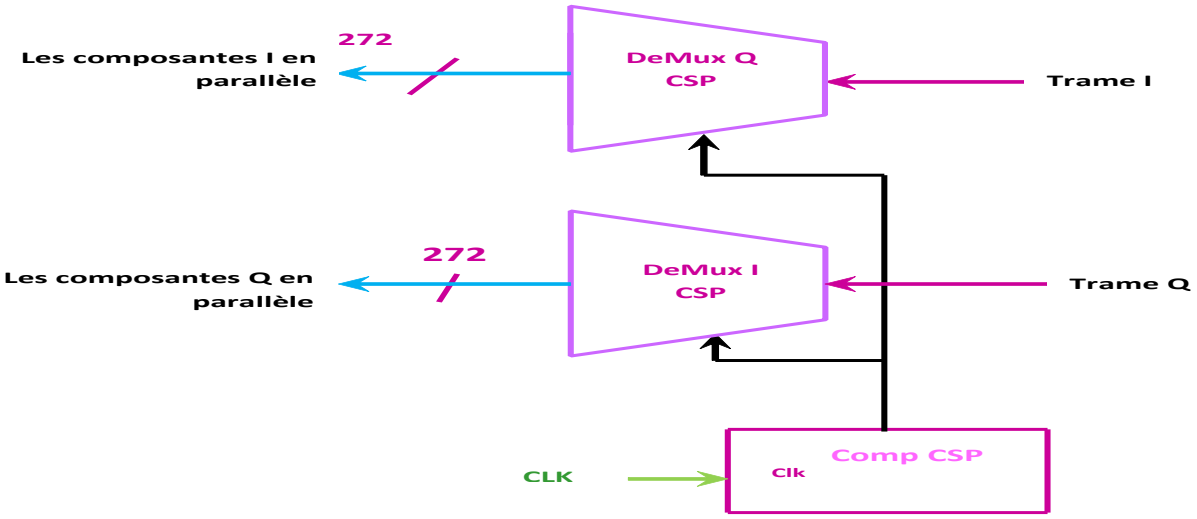

Fig. 3·19: Schéma bloc de al conversion Série Parallèle pour la Réception

3.3.3 Suppression du préfixe cyclique

La figure 3·20 montre l'entité de la suppression de l'intervalle de garde qui a :
– 272 entrées pour les composantes réelles,

– 272 entrées pour les composantes imaginaires,
– 256 pour les composantes réelles,
– 256 sorties pour les composantes imaginaires

On constate l'élimination de 16 symboles entre l'entrée et la sortie représentant l'intervalle de garde.

Fig. 3·20: L'entité de la Suppression de l'Intervalle de Garde

Le schéma bloc de l'entité de la suppression de l'intervalle de garde est illustré par la figure 3·21.Après la conversion série parallèle, les 272 symboles seront sauvegardés dans

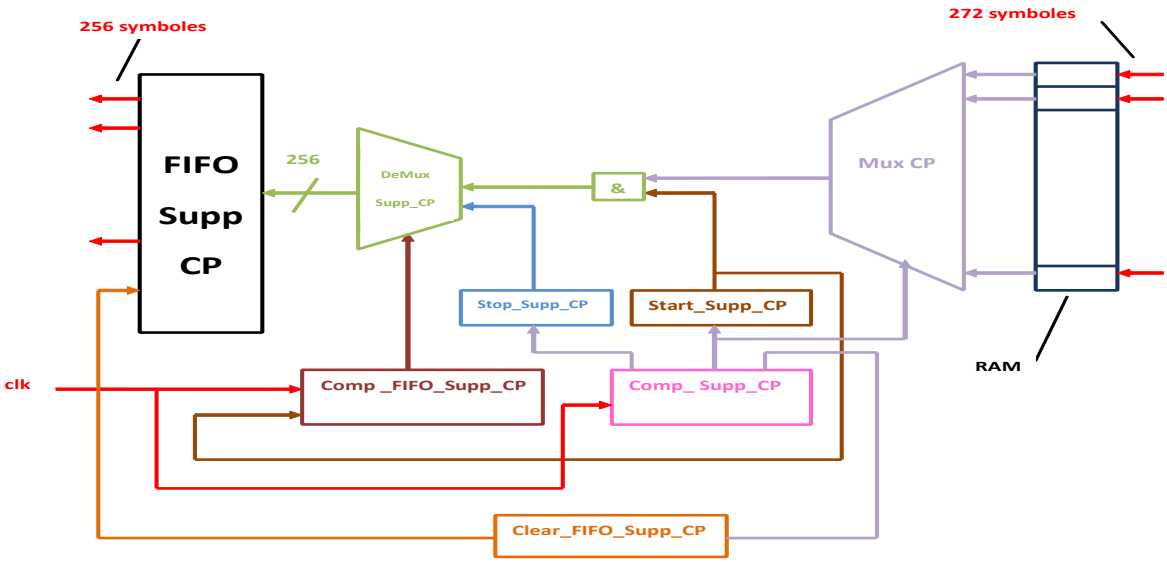

Fig. 3·21: Schéma bloc de la Suppression de l'Intervalle de Garde

une RAM, A chaque cycle d'horloge un symbole de la RAM va passer par le Multiplexeur. Entre temps, le même signal de sélection va passer par la porte AND. Ce dernier sera égal à 0 tant que Start_Supp_CP n'est pas déclenché donc n'est pas à niveau haut (voir Figure 3·21). Ce signal de sélection sera égal à 1 quand le compteur Comp_Supp_CP modulo

Sorties\Entrées	Réel	Imaginaire
Y[0]	(y[0]réel+y[2]réel +y[1]réel+y[3]réel)/4	(y[0]im+y[2]im +y[1]im+y[3]im)/4
Y[1]	(y[0]réel-y[2]réel +y[1]im-y[3]im)/4	(y[0]im-y[2]im -y[1]réel+y[3]réel)/4
Y[2]	(y[0]réel+y[2]réel -y[1]réel-y[3]réel)/4	(y[0]im+y[2]im -y[1]im-y[3]im/)4
Y[3]	(y[0]réel-y[2]réel -y[1]im+y[3]im)/4	(y[0]im-y[2]im +y[1]réel-y[3]réel)/4

Tab. 3.5: Les sortie de la FFT

272 aurait atteint ces 15 cycles (grâce à une fonction logique Start_Supp_CP), c'est à dire que les 16 premiers éléments de la RAM (les symbole de l'intervalle de garde) seront supprimés. A partir du 17ème cycle (16), le signal Start_Supp_CP sera égal à 1 et donc la sortie de la porte AND est la même que le symbole d'information sélectionné à partir de la RAM.

Ce symbole se retrouvera à l'entrée du Demultiplexeur qui va sélectionner à partir du $17^{ème}$ élément de la RAM grâce à la sortie du compteur jusqu'au $272^{ème}$ symbole d'information. Comp_FIFO_Supp_CP modulo 256 qui sera déclenché lui aussi quand le Comp_Supp_CP serait à son $17^{ème}$ cycle (les deux compteur seront synchronisés - même horloge-). Après que les deux compteurs auront terminé leurs boucles, un signal d'arrêt de sélection Stop_Supp_CP sera émis au multiplexeur, les données seront lues par le bloc suivant (celui de la FFT) et elles seront effacées juste après grâce au signal Clear_FIFO_Supp_CP (voir Figure 3·21)

3.3.4 FFT

Avec le même principe développé dans la sous-section IFFT de la section Emetteur, nous avons procédé en prenant le facteur multiplicateur égal à W_N^{-nk}. Après le même développement et pour N=4, nous avons obtenu le tableau 3.5 .Les Y[k] sont les sorties de la FFT qui sont des composantes fréquentielles et les y[k] sont les entrées de la FFT qui sont des composantes temporelles. Nous généralisons pour N=256 en utilisant (comme pour l'IFFT) (64×64) unités à N=4 comportant 4 unités BUTTERFLY.

3.3.5 Estimation des symboles d'information c_k

L'estimation des symboles d'information se fait par égalisation du canal et correction des symboles résultant de la FFT. L'entité Estimation des symboles c_k(Figure 3·22) com-

Fig. 3·22: L'entité Estimation des Symboles

porte à l'entrée 192 composantes réelles I, 192 composantes imaginaires Q car nous avons émis 192 symboles d'information (voir la section IFFT) et les 8 pilotes reçus. A la sortie, nous avons 192 composantes réelles estimées I et 192 composantes imaginaires estimées Q. Le schéma bloc de l'entité de l'estimation des symboles d'information est donné par

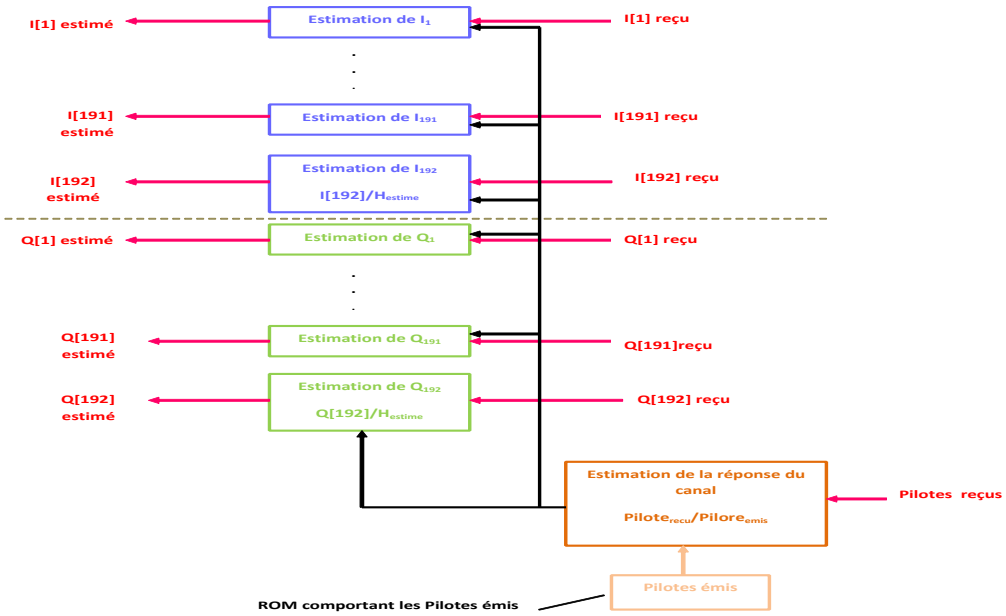

Fig. 3·23: Schéma bloc de l'estimation des symboles

la figure 3·23. Il faut mentionner que ce traitement se fait en parallèle pour les I et les Q. Après l'opération de la FFT, nous récupérons les pilotes dans le domaine fréquentiel. Et puisque nous connaissons les pilotes émis, nous égalisons le canal en calculant sa fonction du transfert (**1**,) donnée par l'équation 3.11

$$H[k]_{estimé} = \frac{Pilote_{recu}}{Pilore_{emis}} \tag{3.11}$$

Le signal reçu s'exprime par (3.12) :

$$Y[k] = H[k]_{estimé}.X[k] \tag{3.12}$$

et donc, le symbole estimé est calculé par l'équation 3.13

$$X[k] = \frac{Y[k]}{H[k]_{estimé}} \tag{3.13}$$

3.3.6 Demmaper

La figure 3·24 montre l'entité Demapper. Nous avons à l'entrée une composante réelle, une autre imaginaire et une horloge. A la sortie, nous avons les 4 bits traduisant le vecteur (I,Q). L'estimateur a (192+192) sorties et par conséquent nous ne pouvons pas relier les

Fig. 3·24: L'entité DeMapper

deux blocs directement. Pour cela, nous avons introduit un bloc qui relie l'Estimateur au Demmaper (figure 3·25). Ce bloc contient deux Mux (192 vers 1) : un pour I et l'autre

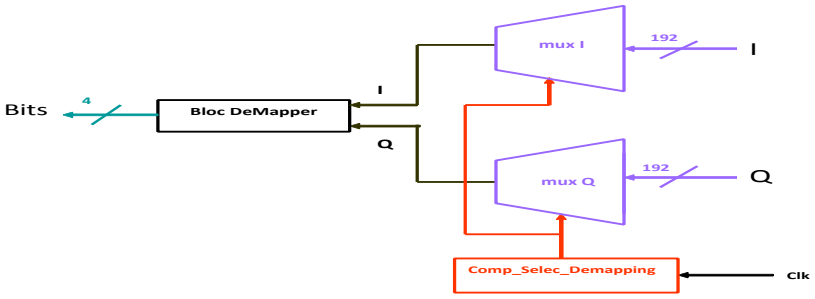

Fig. 3·25: Bloc reliant le Demapper à l'Estimateur

pour Q et qui sont contrôlés par le même compteur de sélection.

Les sorties des deux Mux doivent être synchronisées : (**5**,) :

- I[1] et Q[1] au Premier cycle d'horloge.
- I[2] et Q[2] au 2ème cycle d'horloge.

  .....
- I[192] et Q[192] au 192ème cycle d'horloge.

3.3.7 Conversion Parallèle Série

L'entité convertisseur parallèle/série 4 vers 1 est montrée par la figure 3·26. Elle se compose de 4 bits parallèles et une horloge à l'entrée et une sortie série.

Le schéma détaillé est donné par la figure 3·27.Nous utilisons un Multiplexeur 4 vers 1.

Fig. 3·26: Entité Convertisseur Parallèle Série

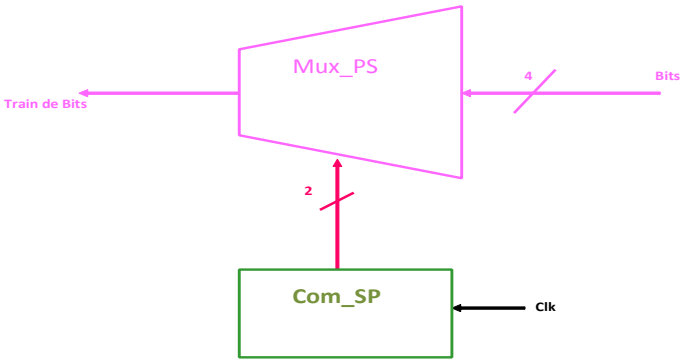

Fig. 3·27: Schéma Bloc du Convertisseur Série Parallèle

Chapitre 4

Synthèse, simulation et implémentation

Une fois le principe de fonctionnement de chaque bloc est établi, nous passons à créer les programmes ou les codes sources VHDL en Xilinx ISE 8.2i. Les programmes validés permettent d'obtenir le schéma hardware de chaque bloc qui sera implémenté et de visualiser les chronogrammes des blocs pour vérifier leur conformité avec l'idée de fonctionnement de chacun de ces blocs développé dans le chapitre **3**.

4.1 Synthèse et Simulation des blocs du système

Nous avons présenté précédemment les schémas bloc détaillés des blocs composant notre système. Notre implémentation se limitera au cas de 4 sous porteuses de cause de la complexité du processeur IFFT/FFT. Dans cette section, notre travail consiste à programmer en *VHDL*, synthétiser avec *ISE 8.2i* et simuler avec le *MODEL SIM 6.1a* les blocs du système séparément. Notre séquence d'information sera subdivisée en bloc de 4bits.

4.1.1 Convertisseur Série Parallèle de l'Emetteur

On subdivise la séquence d'information en blocs de 4 bits chacun. Soit la séquence [....1101.0010]. La conversion série parallèle de ce bloc donne :

$$[....1101.0010] \Longrightarrow \begin{matrix} .. & 1 & 0 \\ .. & 0 & 1 \\ .. & 1 & 0 \\ .. & 1 & 0 \end{matrix}$$

La programmation en VHDL (selon l'idée exposée dans la **sous-section 3.1.1**) sur Xilinx ISE a permis d'obtenir le schéma hardware du convertisseur Série Parallèle de l'émetteur grâce à l'option Synthèse de l'ISE XST. Ce schéma hardware est montré sur la figure 4·1

Sur la figure 4·2,nous observons la parallélisation de la séquence 1101 faite par le simulateur Model Sim :

- 1 sur paralel1_out,
- 0 sur paralel2_out,

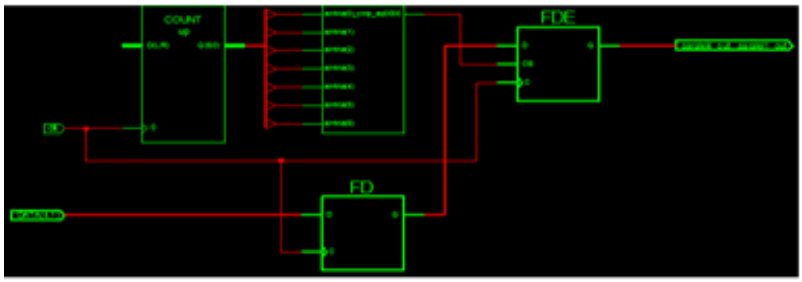

Fig. 4·1: Le schéma hardware du convertisseur Série Parallèle de l'émetteur

– 1 sur paralel3_out,

– 1 sur paralel4_out.

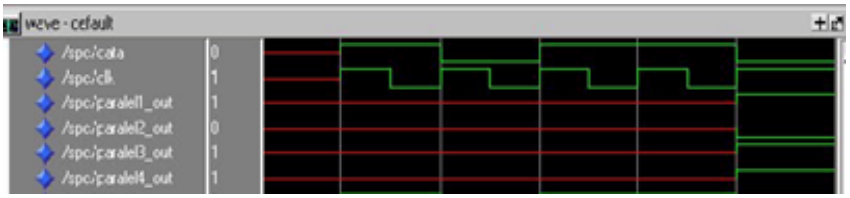

Fig. 4·2: Parallélisation de la séquence 1011

Nous observons dans la Figure 4·3 les chronogrammes de simulation de la parallélisation de la séquence 0010 pour counting=4 : 1 sur paralel1_out,

– 0 sur paralel2_out,
– 1 sur paralel2_out,

– 0 sur paralel3_out,

– 0 sur paralel4_out.

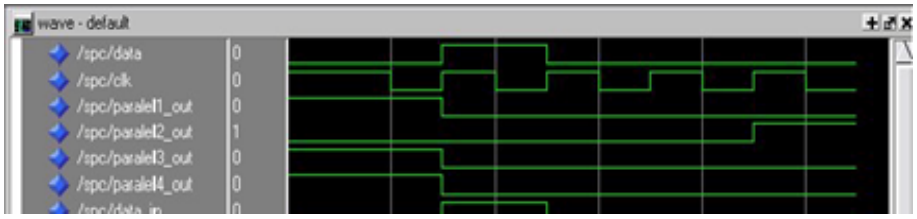

Fig. 4·3: Parallélisation de la séquence 0100

4.1.2 MAPPING

Les blocs se sont appliqués aux entrées du MAPPER un par un. La programmation en VHDL du mapper donnée (**sous-section 3.1.2**) (voir programme mapping en Annexe) et la synthèse du programme conduit au schéma hardware donné par la figure 4·4

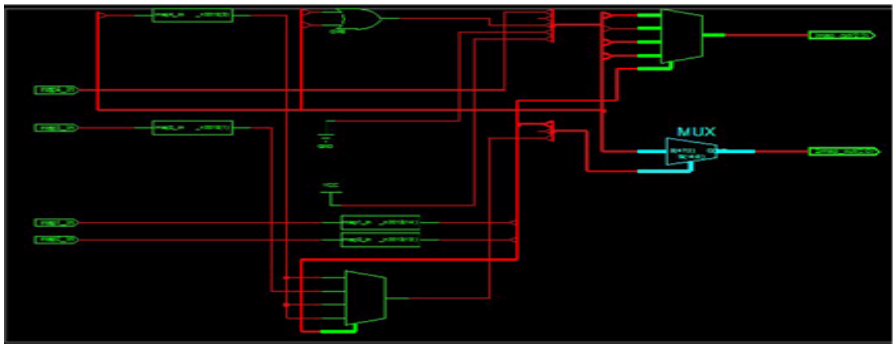

Fig. 4·4: Le schéma hardware du mapper

L'exécution de la simulation du mapper donne les sorties pour le cas 1101 les composantes réelle I=-1 et imaginaire Q=-1 comme le montre la figure 4·5. Pour un autre

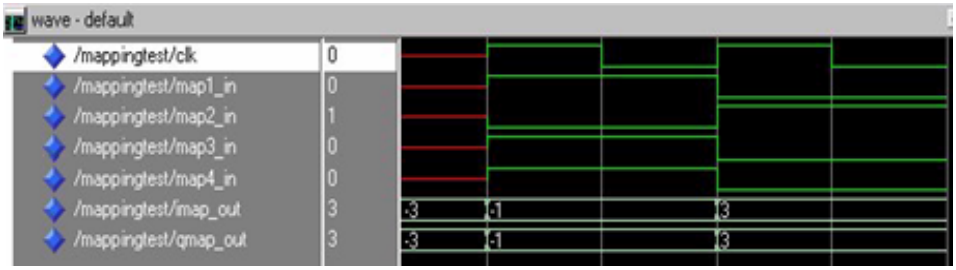

Fig. 4·5: Simulation du Mapping

cas 0010, les sorties sont I=3 et Q=3(figure 4·5). Le mapping se fait pour chaque cycle d'horloge.

4.1.3 IFFT

La programmation du bloc IFFT (**sous-section 3.1.3**) en VHDL et l'utilisation de l'ISE 8.2i pour le cas N=4 donne le schéma de synthèse global (figure 4·6).

L'exécution de la simulation du bloc IFFT par le logiciel MODEL SIM donne les chronogrammes représenetés par la figure 4·7. On observe sur cette dernière figure les signaux d'entrée suivants :

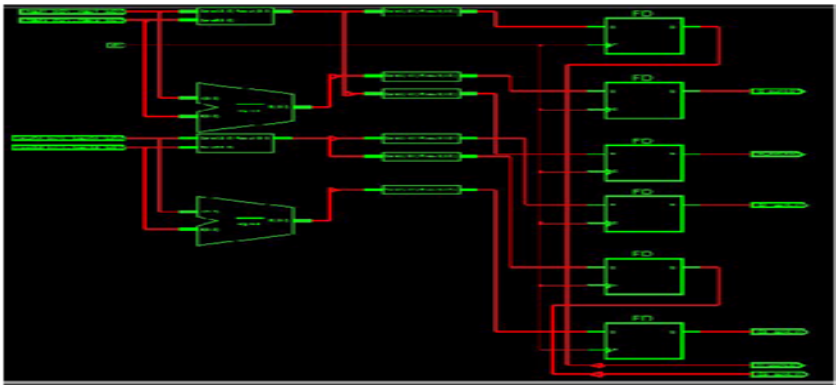

Fig. 4·6: Le schéma hardware de l'IFFT pour N=4

- interi1_in et interi3_in sont les deux composantes réelles des deux séquences binaires 1011 et 0100 qui sont égales respectivement à -1 et 3.
- interq1_in et interq3_in sont les deux composantes imaginaires des deux séquences binaires 1011 et 0100 qui sont égales respectivement à -1 et 3.
- Nous rappelons que pour interi2_in et interq2_in qui sont les composantes de la sous porteuse non utilisée sont nuls.
- interi4_in et interq4_in sont égales respectivement à 2 et 2 et qui sont les composantes du pilote.

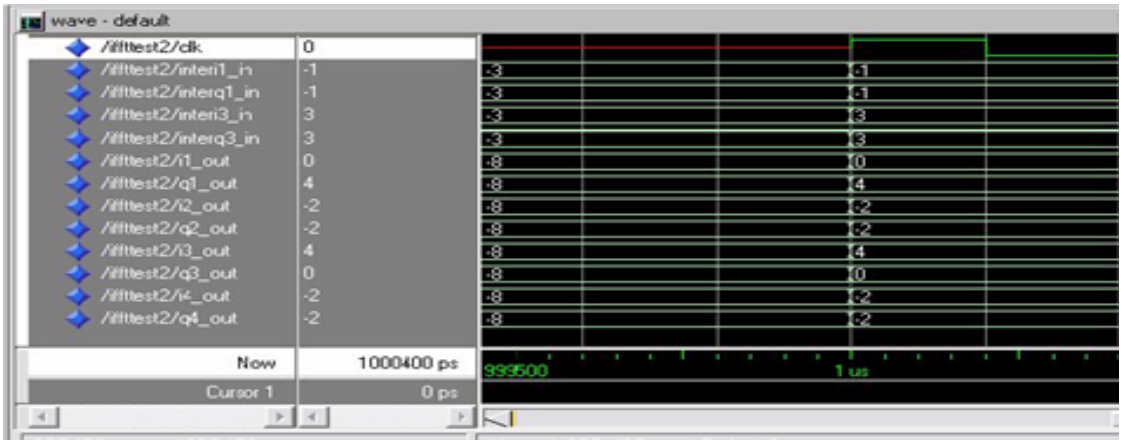

Fig. 4·7: Simulation de l'IFFT pour N=4

Les signaux de sortie sont :

- i1_out et q1_out sont les composantes de la première composante temporelle.
- i2_out et q2_out sont les composantes de la deuxième composante temporelle.
- i3_out et q3_out sont les composantes de la troisième composante temporelle.
- i4_out et q4_out sont les composantes de la quatrième composante temporelle.

IN	1	2	3	4	OUT	1	2	3	4
I	-1	0	3	-2	I	0	-2	4	-2
Q	-1	0	3	2	Q	4	-2	0	-2

Tab. 4.1: Résultats de simulation de l'IFFT

IN	1 pos	2 pos	3 pos	4 pos	OUT	1 pos	2 pos	3 pos	4 pos	5 pos
I	-1	0	3	-2	I	2-	0	-2	4	-2
Q	-1	0	3	2	Q	-2	4	-2	0	-2

Tab. 4.2: Résultats de simulation du bloc de l'insersion de l'intervalle de garde

Les résultats de la simulation du bloc de l'IFFT sont représentés dans le tableau 4.1

4.1.4 Ajout de l'intervalle de Garde

Le schéma de la synthèse du bloc de l'insertion de l'intervalle de garde représenté par la figure 4·8, obtenu par la programmation en VHDL du bloc de l'insertion de l'intervalle de garde (**sous-section 3.1.4**) et la synthèse par ISE XST.

Puisque nous utilisons une IFFT pour N=4, notre rapport cyclique est représenté par un seul échantillon (le quart des sous porteuses de l'IFFT).

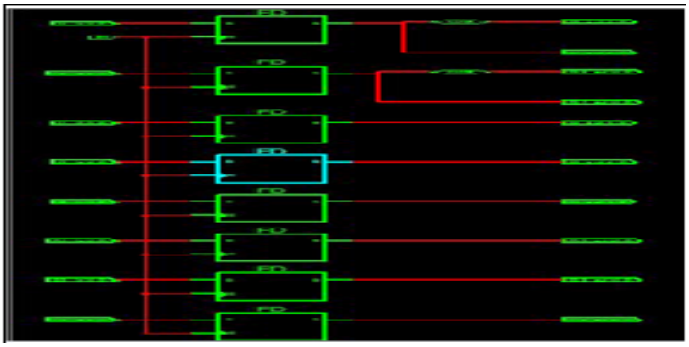

Fig. 4·8: Simulation du bloc de l'ajout du préfixe cyclique

La simulation de ce bloc par le MODEL SIM permet d'obtenir les chronogrammes de l'ajout de l'intervalle de préfixe cyclique (figure 4·9).

Nous constatons la dernière composante (I,Q) se plaçant à la première position du vecteur de sortie. Le tableau 4.2 résume les E/S de ce bloc.

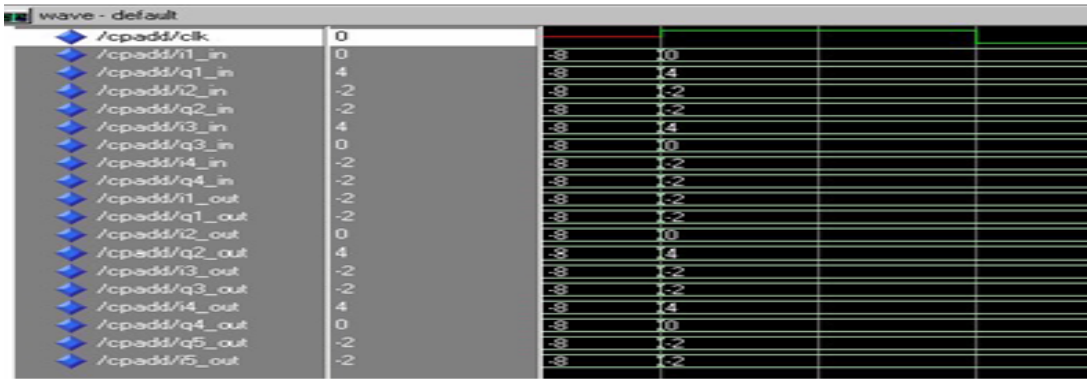

Fig. 4·9: Simulation du bloc de l'insertion de l'intervalle de garde

4.1.5 Conversion Parallèle Série de l'Emetteur

La programmation du bloc CPS de l'émetteur en VHDL et sa simulation conduit au chronogramme de la figure 4·10.

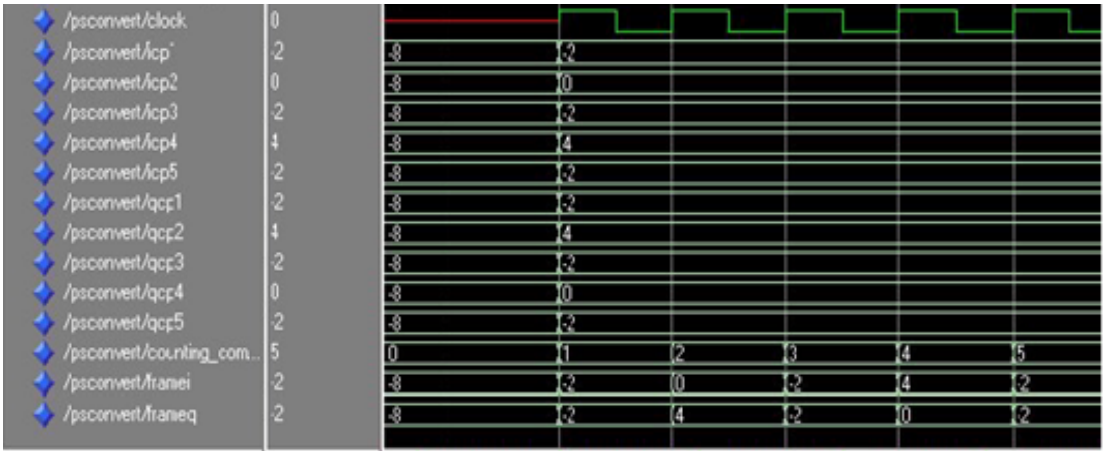

Fig. 4·10: Simulation de la conversion parallèle série de l'émetteur

Au premier cycle d'horloge (counting =1), la première composante réelle va se placer à la sortie framei et la première composante imaginaire va se placer sur frameq. Au deuxième cycle d'horloge, ce sera le tour de la deuxième composante ainsi de suite....On observe sur le tableau 4.3 les deux séquences en série.

Le schéma de la synthèse de ce bloc s'étend sur 5 fenêtres de l'ISE, nous avons préféré ne pas le mettre dans le rapport.

compteur de sélection	1	2	3	4	5
framei (séquence réelle)	-2	0	-2	4	-2
frameq (séquence imaginaire)	-2	4	-2	0	-2

Tab. 4.3: Résultats de simulation du convertisseur parallèle série de l'émetteur

4.1.6 Modulation Radio Fréquence

Après avoir programmé (**sous-section 3.1.6**) et synthétisé le bloc de la modulation RF, nous avons obtenu le schéma hardware représenté sur la figure 4·11. Le MODEL SIM a permis de visualiser la modulation QAM analogique représentée sur la figure (4·12).

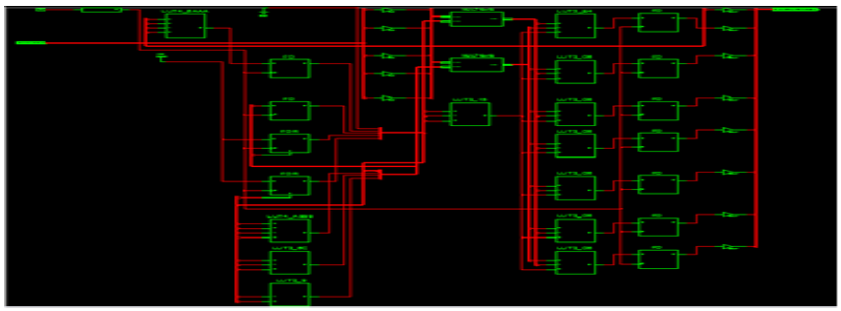

Fig. 4·11: Le schéma hardware du modulateur RF

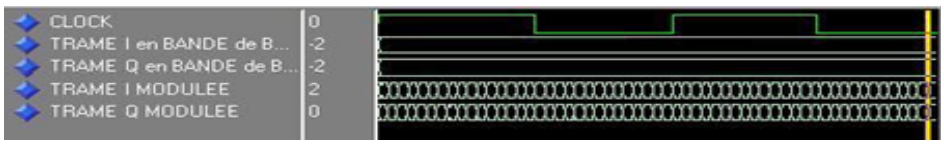

Fig. 4·12: Simulation du modulateur RF

4.1.7 Canal

La programmation et la synthèse du bloc du canal donne le schéma hardware de la figure (4·13)

Le résultat de la simulation donné par le MODEL SIM est donné par la figure 4·14.

On observe les versions retardées d'une séquence émise dans un canal à trajet multiple : -2 0 4 8 6 -3 5 -7 -5 et on constate que :

- Le déclenchement du SIGNAL_WITHOUT_DELAY est fait au même temps que l'émission du signal modulé.

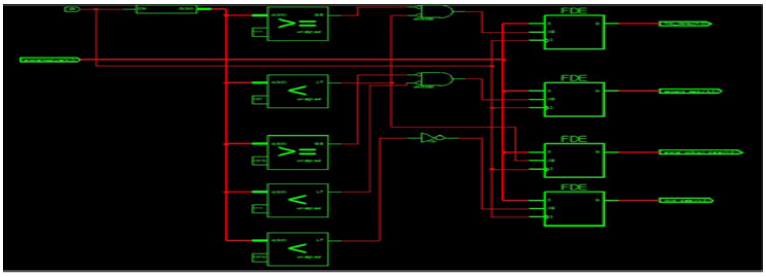

Fig. 4·13: Le schéma hardware des versions retardées

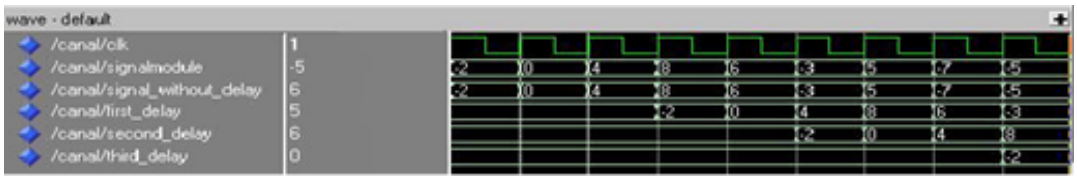

Fig. 4·14: Simulation de l'effet multitrajet du canal

- Le déclenchement du signal FIRST_DELAY est fait au 4ème cycle d'horloge et la séquence n'a pas changé.
- Le déclenchement pour le signal SECOND_DELAY au 6ème cycle d'horloge.
- Le déclenchement pour le signal THIRD_DELAY au 10ème cycle d'horloge (nous n'avons pas prendre toute la séquence sur les retards à cause de la limite de la fenêtre wave).

4.1.8 Le démodulateur Radio Fréquence

Le schéma hardware du démodulateur obtenu après avoir programmé le bloc suivant l'idée exposée dans la section est montré sur la figure 4·15. La simulation de la démodulation d'une séquence du signal reçu portant le symbole 4 est montrée sur la figure 4·16

4.1.9 La conversion Série Parallèle du Récepteur

Le schéma hardware de la conversion série parallèle du récepteur est montré sur la figure 4·17 en se basant sur le principe de la section (voir le programme en annexe).

Rappelons que la conversion Série Parallèle au niveau du récepteur a été faite simultanément pour la composante I et la composante Q (voir figure 4·18)

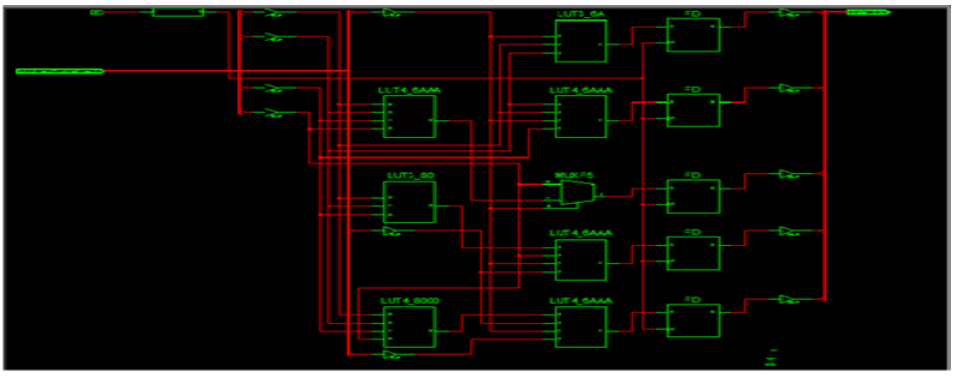

Fig. 4·15: Schéma hardware du démodulateur

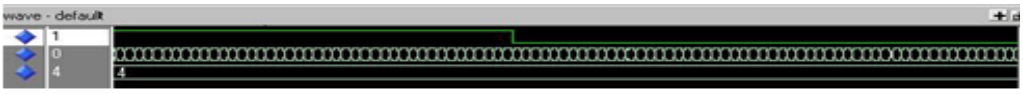

Fig. 4·16: Démodulation d'une composante réelle égale à 4

4.1.10 La suppression de l'intervalle de garde

Le schéma hardware du bloc de suppression de l'intervalle de garde est sur la figure 4·19.

Sur la figure 4·20, nous observons que la première composante temporelle (i_paralel1_in et q_paralel1_in) a été supprimée à la sortie du bloc

4.1.11 FFT

Le schéma hardware de la FFT est sur la figure 4·21.
La figure 4·22 montre la simulation de la FFT.
Le tableau 4.4 donne le résultat de la FFT

- A la quatrième position, nous récupérons le pilote (2,-2).
- A la deuxième, les zéros (0,0) .
- Et aux deux autres positions, les données (-1,-1) et (3,3).

In	1	2	3	4	Out	1	2	3	4
Re	0	-2	4	-2	Re	-1	0	3	-2
Im	4	-2	0	-2	Im	-1	0	3	2

Tab. 4.4: Résultats de simulation dde la FFT

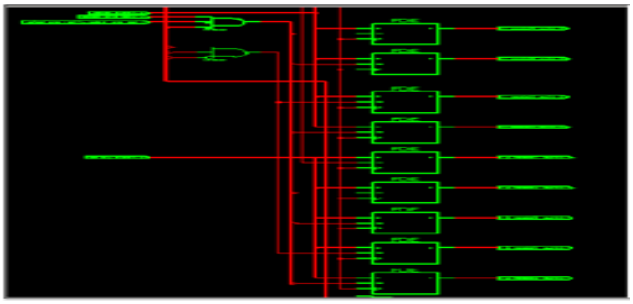

Fig. 4·17: Le schéma hardware du convertisseur série parallèle

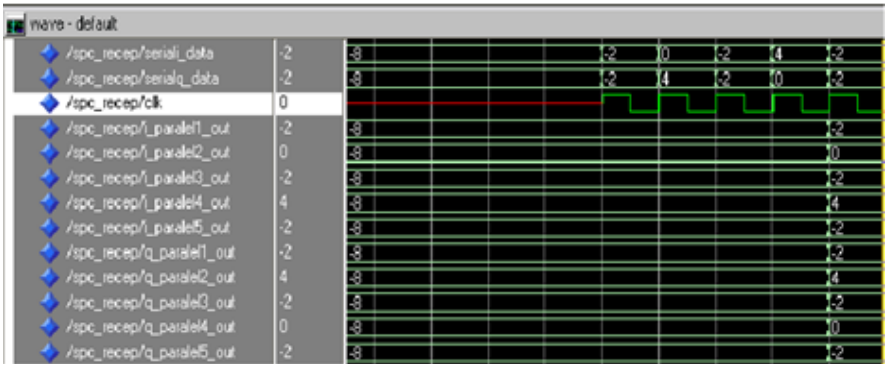

Fig. 4·18: La simulation de la conversion série parallèle du récepteur

4.1.12 DEMAPPING

Le schéma hardware du Demapping est montré sur la figure 4·23.

Les vecteurs d'entrées sont (-1, -1) et (3, 3) qui vont passer par le DEMAPPER. (voir figure 4·24)

Le tableau 4.5 donne le résultat de la simulation du Demapper.

4.1.13 Conversion Parallèle Série du Récepteur

Le schéma hardware du convertisseur parallèle série sur la figure 4·25.

Nous pouvons voir les deux séquences (1011et0010) sur la sortie finalsignal.(figure 4·26)

I (imap_in)	Q (qmap_in)	map1_out	map2_out	map3_out	map4_out
-1	-1	1	0	1	1
3	3	0	1	0	0

Tab. 4.5: Résultats de simulation du Demapper

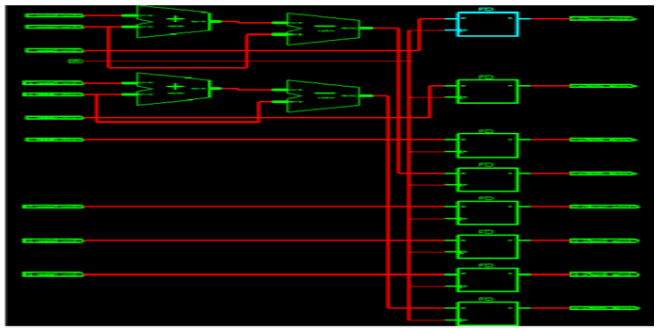

Fig. 4·19: Le schéma hardware bloc de la suppression de l'intervalle de garde

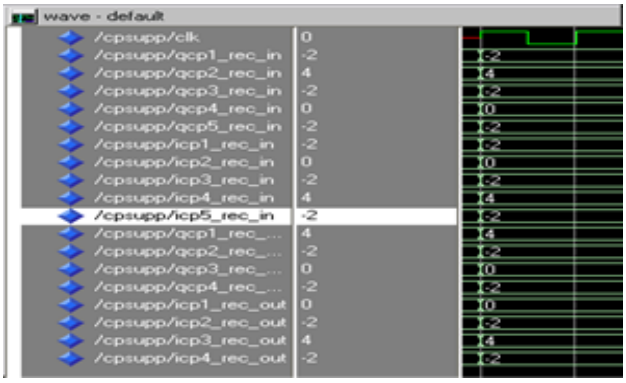

Fig. 4·20: Simulation de la suppression de l'intervalle de garde

La séquence 1011 résulte des entrées pour :
- ps1_in <=1,
- ps2_in <=0,
- ps3_in <=1,
- ps4_in <=1.

La séquence 1011 résulte des entrées pour :
- ps1_in <=0,
- ps2_in <=0,
- ps3_in <=1,
- ps4_in <=0.

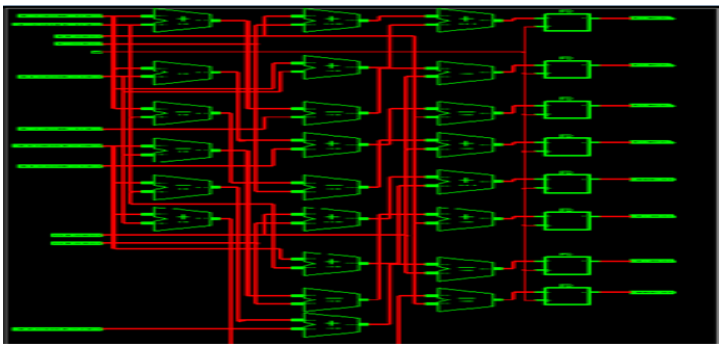

Fig. 4·21: Le schéma hardware bloc de la FFT N=4

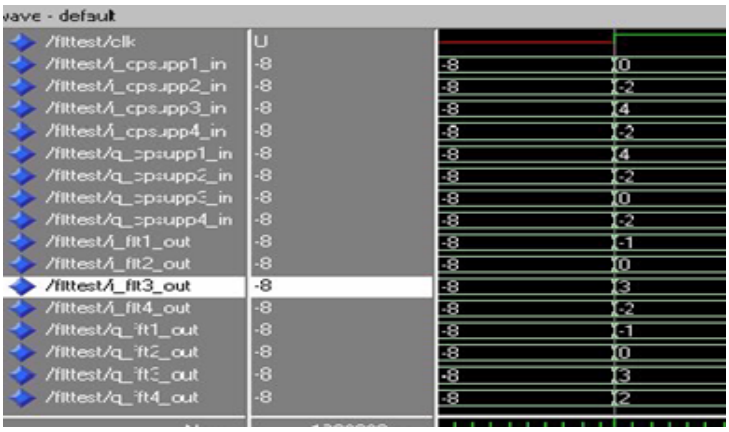

Fig. 4·22: Simulation du bloc FFT

4.2 Implémentation du système OFDM

Dans ce chapitre, nous regroupons tous les programmes des blocs détaillés dans le chapitre dans un seul projet principal ISE (OFDM.ise). La synthèse et la simulation du système y sont traitées. La synchronisation et la coordination entre les différents blocs sont très importantes dans cette étape. Les opérations importantes exécutées par le logiciel ISE sont les suivantes :

- La synthèse du projet OFDM.ise,
- L'appel du logiciel MODEL SIM pour la simulation,
- Le mapping du système,
- Le routage et interconnexion entre les différents blocs,
- Et implémentation du système OFDM.

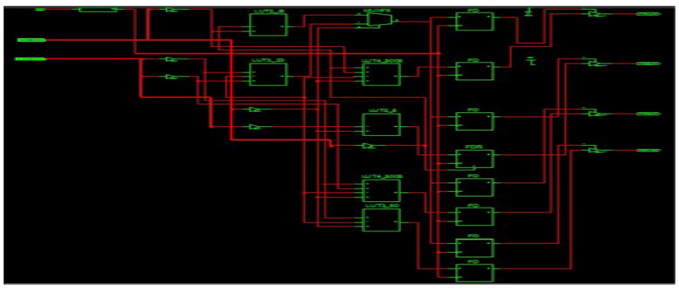

Fig. 4·23: Le schéma hardware bloc du Demapper

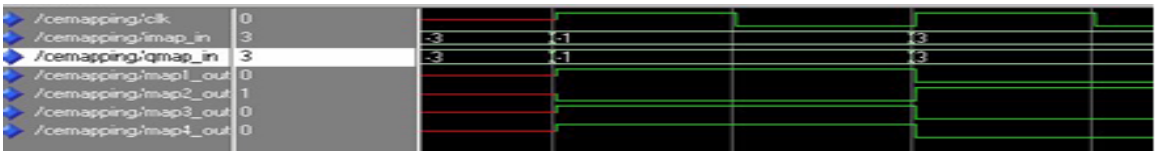

Fig. 4·24: Simulation du Demapper

4.2.1 Compilation

La compilation de notre projet par l'ISE (voir Annexe) permet d'exécuter toutes les fonctionnalités cités précédemment (figure 4·27).

4.2.2 Synthèse

Après la compilation, on synthétise le projet OFDM.ise en obtenant l'entité de ce système (voir figure 4·28).

On constate sur cette figure une entrée vectorielle DATA de 8 bits et une horloge clk_in synchronisant le traitement. Les sorties aff1_data_out et aff2_data_out sont deux afficheurs 7 segments se trouvant sur la carte FPGA VIRTEX II pour visualiser la séquence envoyée. Le schéma détaillé de cette entité est donné par la figure (4·29).Pour des raisons de résolution de la limite de la fenêtre de l'ISE, nous avons zoomé la partie de l'émetteur à partir du schéma de synthèse du système (voir figure 4·30) .

De la même manière, nous avons procédé pour le canal (figure 4·31) et le récepteur (figure 4·32).

4.2.3 Simulation

Après avoir exécuté la compilation et la synthèse de notre système, nous procédons à sa simulation par le MODEL SIM. Nous obtenons le résultat de la simulation pour la

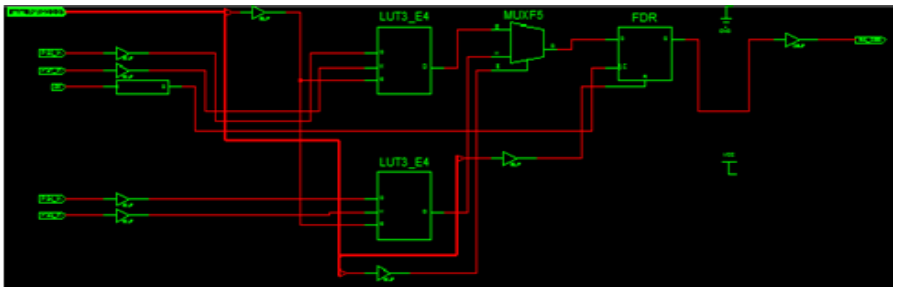

Fig. 4·25: Le schéma hardware du convertisseur parallèle série du récepteur

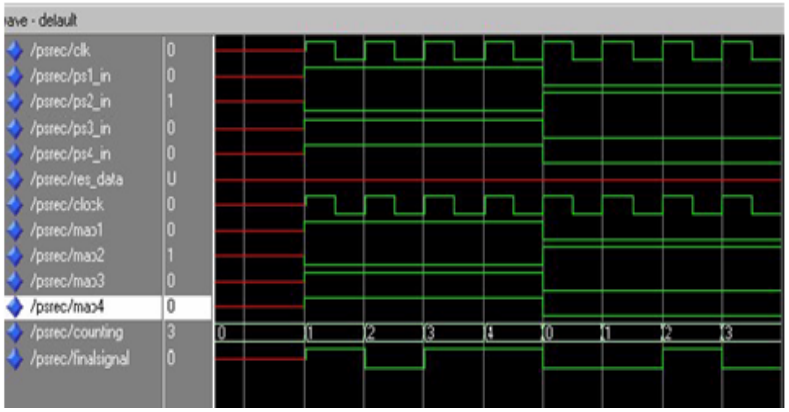

Fig. 4·26: Simulation du convertisseur parallèle série du récepteur

séquence 10110100 représentée par une fenêtre Model Sim. Faute d'affichage, nous avons découpé cette fenêtre en plusieurs figures :

- La figure 4·33 représente la parallélisation de la séquence envoyée 10110100,
- La figure 4·34 représente le mapping et l'IFFT. Nous pouvons voir la correspondance du vecteur 0100 à(-3, 1) et 1011 à (1,-3). Après un calcul effectué par l'IFFT, nous avons obtenu les composantes temporelles suivante : (0,0) ; (-2,2) ; (-4,-4) ; (-6,6).
- La figure 4·35 : insertion de l'intervalle de garde et sérialisation.
- La figure 4·36 : les versions retardées du signal émis et la sortie du canal où nous observons que le premier symbole de la rame réelle et celui de la trame imaginaire (le préfixe cyclique) sont déformés dû à l'effet multitrajet (Envoyé : -6,6. Reçu : -4,8).
- La figure 4·37 : démodulation, parallèlisation au niveau du récepteur et suppression de l'intervalle de garde (le symbole déformé).
- La figure 4·38 : FFT, Demapping, sérialisation et récupération de la séquence envoyée 10110100.

La figure 4·39 montre les signaux de contrôle des deux afficheurs 7 segments. Un décodeur va contrôler les deux afficheurs pour afficher la séquence 10110100 qui équivaut en

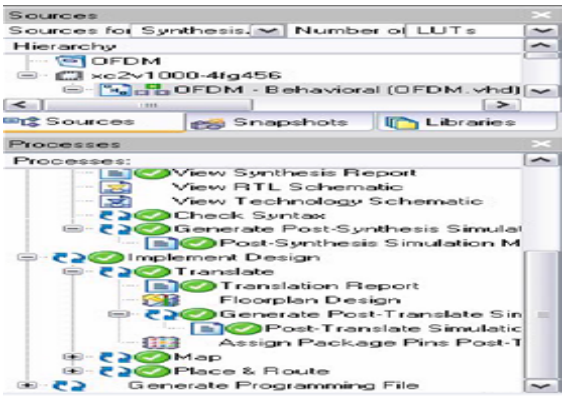

Fig. 4·27: Toutes les fonctionnalités établies

hexadécimal à B4 [$(B)_{16}=(0011111)_2$ sur l'afficheur aff2_data_out et $(4)_{16}=(0100)_2$ sur l'afficheur aff1_data_out].

4.2.4 Mapping

L'utilisation du mapping permet de visualiser l'emplacement de notre système dans la carte VIRTEX II (Figure 4·40). Les points bleus sur ce map montrent les éléments occupés dans la carte.

En zoomant une partie du map, nous visualisons un certain nombre de blocs interconnectés.

Comme exemple, nous observons sur une partie zoomée (la figure 4·41) les connexions entre le bloc FFT (à gauche) et le bloc de la suppression de l'intervalle de garde (à droite). De même nous visualisons sur la figure 4·42 les blocs CSP à gauche, le mapper au milieu et l'IFFT à droite.

4.2.5 Routage

Le routage consiste à relier les différents blocs du système grâce aux signaux intermédiaires assignés dans les codes sources VHDL du système. La figure 4·43 représente les interconnexions entre les blocs de notre système implémenté.

4.2.6 Implémentation

L'opération de l'implémentation de notre système consiste à réaliser physiquement la simulation faite précédemment sur la carte VIRTEX-II. Elle permet également la visualisation du signal de sortie soit sur deux afficheurs 7 segments soit sur un oscilloscope.

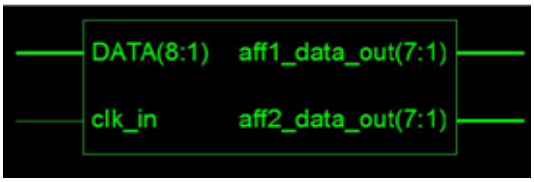

Fig. 4·28: L'entité du système OFDM

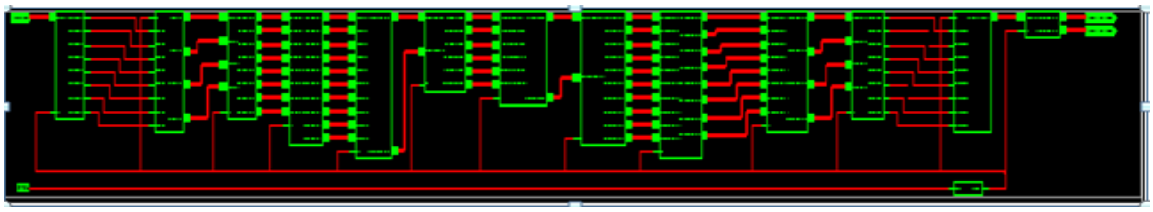

Fig. 4·29: La synthèse du système à implémenter

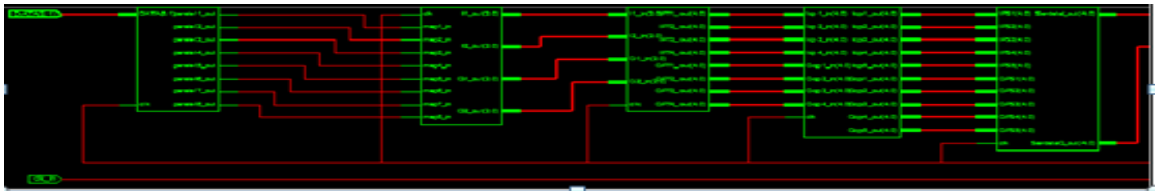

Fig. 4·30: L'émetteur

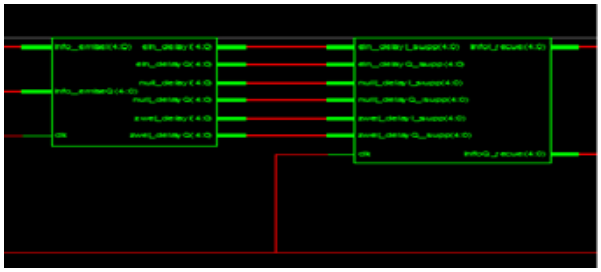

Fig. 4·31: Le canal

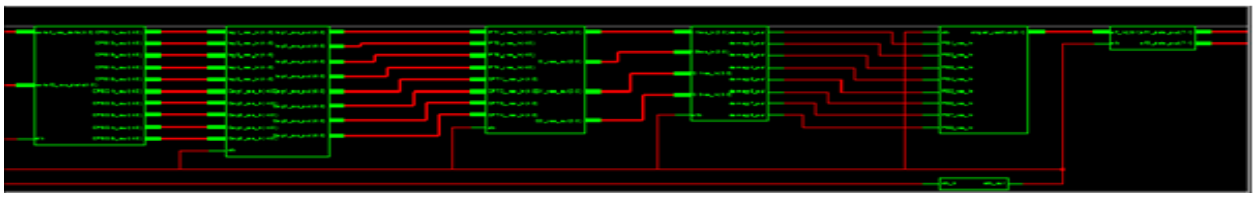

Fig. 4·32: Le récepteur

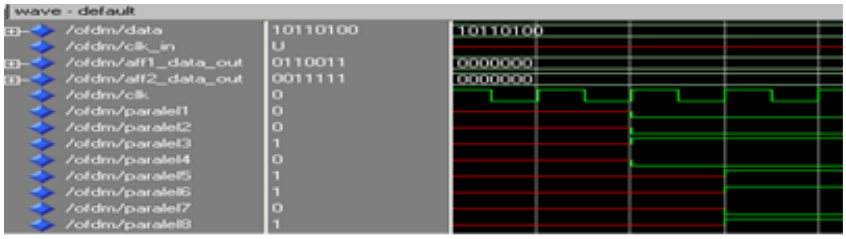

Fig. 4·33: L'envoi de la séquence et parallélisation

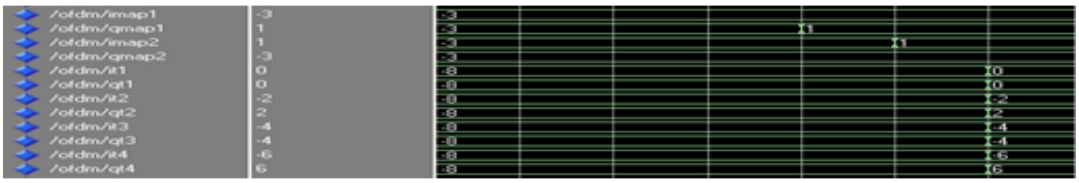

Fig. 4·34: mapping et calcul des composantes temporelles

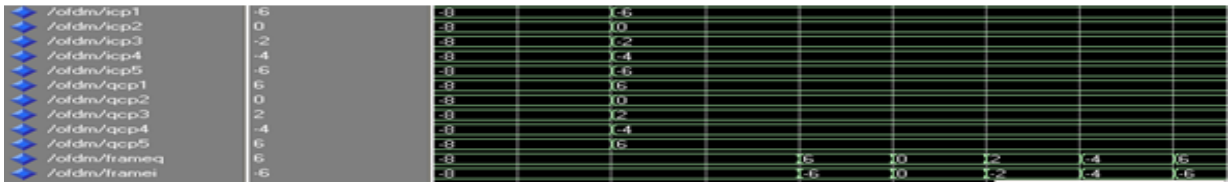

Fig. 4·35: Ajout de l'intervalle de garde et sérialisation

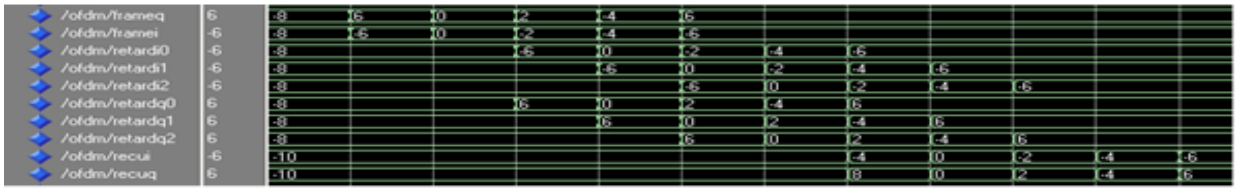

Fig. 4·36: L'effet multitrajets du canal

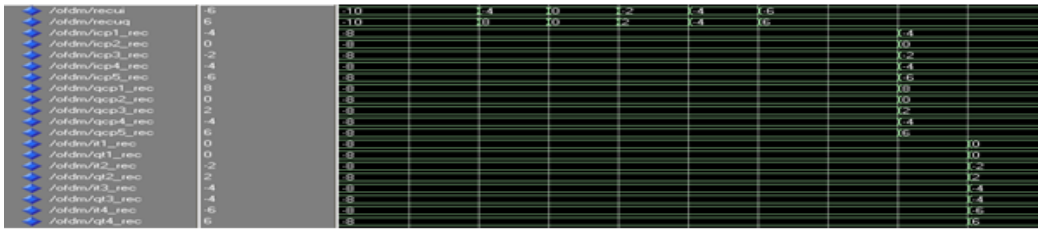

Fig. 4·37: Parallélisation au niveau du récepteur et suppression de l'intervalle de garde

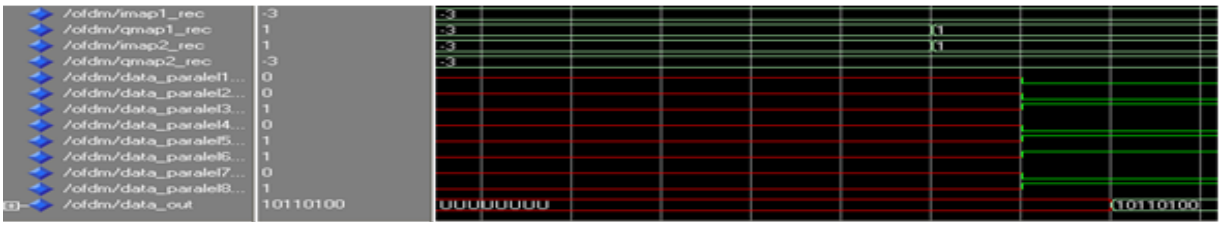

Fig. 4·38: FFT, demapping et sérialisation et récupération de la séquence envoyée

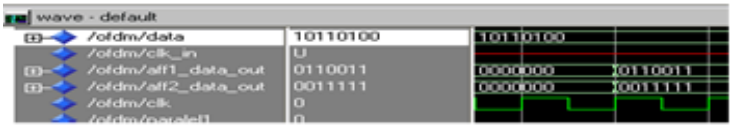

Fig. 4·39: Affichage sur deux afficheurs 7 segments

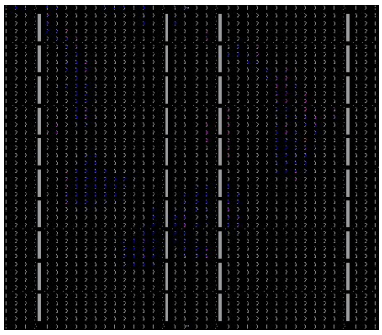

Fig. 4·40: Mapping de la carte VIRTEX II

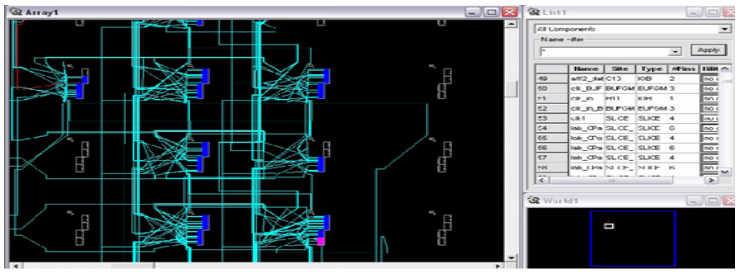

Fig. 4·41: FFT et le bloc de la suppression de l'intervalle de garde

Assignement des PINS

Cette étape permet de placer les entrées et les sorties du bloc à implémenter dans la carte en respectant les tables jointes avec la carte VIRTEX II .

Le tableau de l'assignement est donnée par la figure 4·44.

La figure 4·45 montre la fenêtre de l'assignement des entrées/sorties de notre système sur notre carte FPGA. Nous observons sur cette figure :

- L'hexagone vert représente l'horloge à 24 MHZ.
- Les cercles roses représentent les entrées (les 8 bits).
- Les cercles jaunes sont l'emplacement des segments de l'afficheur 1 et les bleus de l'afficheur 2.

Réalisation physique

Cette étape consiste à générer un fichier OFDM.bit grâce auquel le projet OFDM.ise est imprimé sur la carte FPGA.

Ce fichier est généré en cliquant deux fois sur "GENERATE PROGRAMMING FILE". On lance le logiciel d'implémentation appelé XILINX Impact qui fait appel à ce fichier

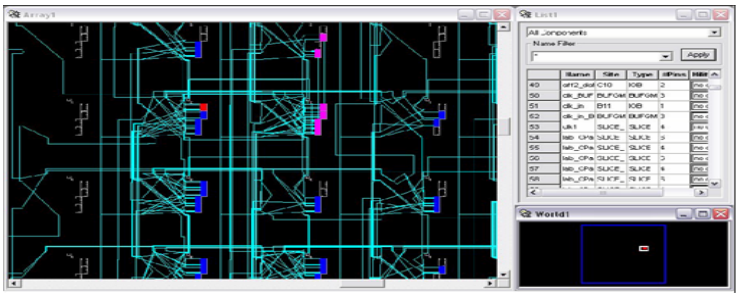

Fig. 4·42: CSP , le mapper et l'IFFT

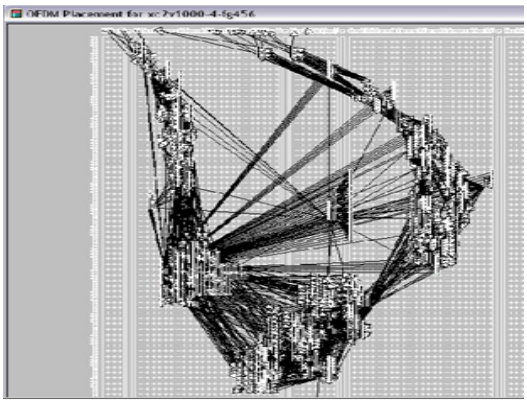

Fig. 4·43: Routage

pour implémenter notre système OFDM sur la carte FPGA VIRTEX II. Une notification de réussite d'implémentation sera affichée à la fin de processus.

Nous pouvons visualiser sur la figure 4·46 l'affichage de la séquence réçue qui est la même celle émise : $(11111111)_2$=$(FF)_{16}$.

4.2.7 Résultats

Une fois notre système est implémenté sur la carte le logiciel ISE génère un rapport donnant des informations sur :

- L'occupation de la carte par le système,
- Les éléments utilisés de la carte,
- La puissance consommée par le système,
- La taille de la mémoire utilisée,
- Le temps d'exécution,

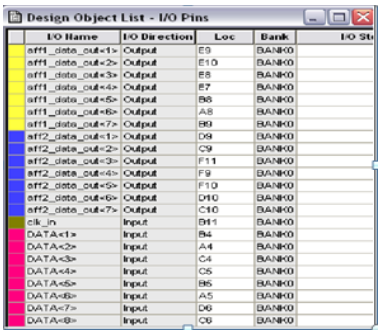

Fig. 4·44: Tableau de l'assignement

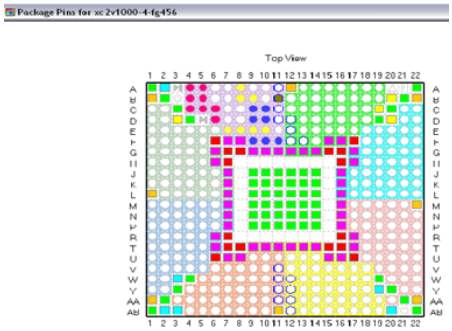

Fig. 4·45: L'assignement des PINS sur la carte

– etc...
Un affichage donne : Peak Memory Usage : 156 MB

La figure 4·47 affiché par l'ISE est celle de l'occupation de notre système. On constate que cette occupation est de 21,71% de la carte. Par conséquent, nous avons la possibilité de compliquer notre système en ajoutant d'autres blocs afin d'améliorer les performances du système en utilisant par exemple des QAM plus volumineuses (64, 256...), une IFFT/FFT à plus de 4 sous porteuses (8, 16, 64, 256,1024...) et un canal radio mobile plus complexe en introduisant d'autres versions retardées, un étalement spectral.....

L'option du Xilinx Power de l'ISE affiche le bilan de consommation de la puissance du système OFDM (Figure 4·48)On remarque dans cette figure qu'à la température ambiante ($25°C$), la puissance consommée est de 333.30 mW, ceci est acceptable de fait que la puissance totale consommée par la carte ne doit pas dépasser 500 mW.

Macro blocs

Le logiciel affiche un rapport donnant le nombre de macro blocs utilisés par le système :
– ROMs : 6

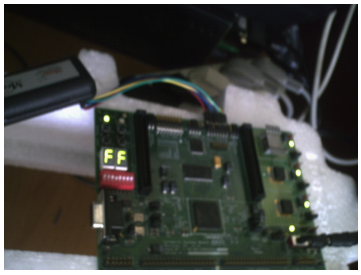

Fig. 4·46: Affichage de la séquence reçue

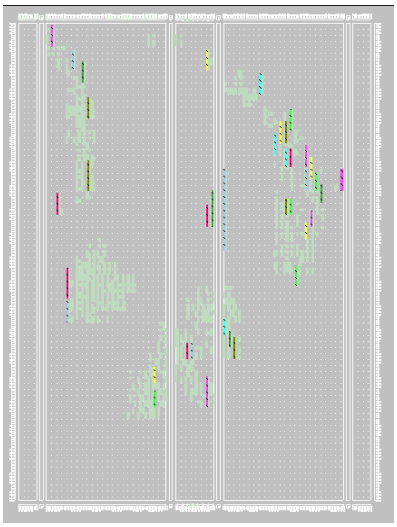

Fig. 4·47: Occupation du système dans la carte VIRTEX II

- 43 16x7-bit ROM : 2
  - 4x3-bit ROM : 4
- Adders/Subtractors :
  - 27-bit adder : 1
    - 3-bit adder : 4
    - 4-bit adder : 2
    - 4-bit subtractor : 4
    - 5-bit adder : 8
    - 5-bit addsub : 4
    - 5-bit subtractor : 4
    - 6-bit adder : 2
    - 6-bit subtractor : 2

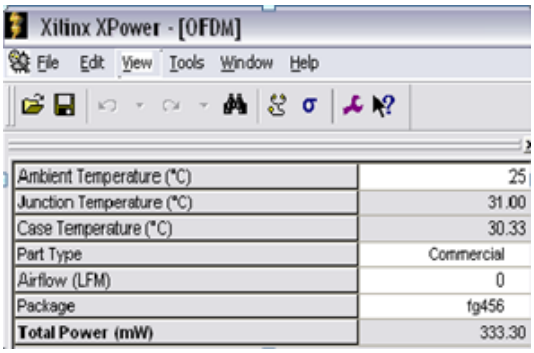

Fig. 4·48: Le bilan de la puissance consommée

  - 7-bit adder : 10
  - 7-bit subtractor : 2
- Counters : 16
  - 10-bit up counter : 3
    - 7-bit up counter : 13
- Registers : 593
  Flip-Flops : 593
- Comparators : 1
  27-bit comparator greatequal : 1
- Multiplexers : 10
  5-bit 4-to-1 multiplexer : 10

Un rapport de statistique du design est donné par l'ISE :
- IOs : 23
- BELS : 1319
- GND : 1
- INV : 22
- LUT1 : 128
- LUT2 : 128
- LUT2_D : 1
- LUT2_L : 10
- LUT3 : 123
- LUT3_D : 5
- LUT3_L : 9
- LUT4 : 359
- LUT4_D : 11
- LUT4_L : 38
- MUXCY : 240
- MUXF5 : 14

- VCC : 1
- XORCY : 229
- FlipFlops/Latches : 680
- FD : 140
- FDE : 513
- FDR : 27
- Clock Buffers : 2
- BUFG : 1
- BUFGP : 1
- IO Buffers : 22
- IBUF : 8
- OBUF : 14

Le dernier rapport donné par l'ISE est le Timing Summary donnant la fréquence maximale à utiliser, le retard minimal du signal d'entrée et le retard maximal du signal de sortie exigé :

- Minimum period : 11.425ns (Maximum Frequency : 87.527MHz)
- Minimum input arrival time before clock : 1.712ns
- Maximum output required time after clock : 5.446ns

4.3 Conclusion

Lors de ce chapitre, nous avons étudié en détail chaque bloc constituant l'émetteur et le récepteur OFDM ainsi que le principal inconvénient d'un canal qui est le multitrajet. Nous avons donné le schéma détaillé de chaque bloc ainsi que la schéma global de notre système. Nous avons développé le code source VHDL de chacun de ces blocs sur le XILINX ISE 8.2i ensuite nous avons regroupé ces codes sources en un seul projet OFDM.ise.

Pour remédier au problème de synchronisation entre les blocs, nous avons inséré un signal qui déclenche le fonctionnement chaque bloc au moment voulu (après que le bloc précédent aurait terminé le traitement). Nous avons simulé chaque bloc ainsi tout le système global. Des résultats ont été obtenus. Grâce à la synthèse, nous avons pu visualiser les blocs du système connectés ainsi que la visualisation le schéma de chaque bloc.

Nous avons implémenté notre système après avoir assigné les entrées et les sorties aux pins de la carte VIRTEX II ce qui nous a permis d'observer les connexions entre les blocs et l'occupation du système dans la carte.

Conclusions générales

L'objectif du travail était l'implémentation du système ODFM. Dans notre travail, nous avons tout d'abord donné un aperçu sur les principes de base de l'OFDM. Nous avons donné le modèle de réalisation à implémenter sur la carte FPGA VIRTEX II, en utilisant le logiciel de développement ISE 8.2i dont la procédure de développement a été présentée en détail dans le chapitre 2.

Pour la réalisation de notre travail, nous avons mis au point une architecture appropriée, tout en se conformant aux spécificités de la norme IEEE 802.16d utilisée dans le WMAN mettant à profit la technique OFDM. Notre architecture comporte la conception de chaque bloc du système. Cette conception consistait à concevoir des programmes en VHDL que nous avons synthétisés et simulés à l'aide de l'outil de conception de XILINX. Par la suite, nous avons procédé à l'assemblage de tous ces programmes en un seul projet ISE. La synchronisation et la coordination entre ces différents blocs étaient des problèmes sérieux à surmonter dans notre projet. Pour remédier à ces problèmes noue avons introduit un signal de déclenchement permettant de déclencher au moment opportun l'exécution de chaque bloc. Une fois ces problèmes résolus, nous avons procédé à la synthèse, la simulation du système qui a validé l'architecture étudiée.

Une fois que la synthèse, la simulation, le routage et le mapping de notre système étaient fait, nous avons procédé à son implémentation qui ont permis la validation de notre architecture. Une fois l'architecture était validée, nous avons implémenté notre système sur la carte de XILINIX VIRTEX II. Nous avons pu vérifié le bon fonctionnement du système en visualisant la séquence reçue sur les afficheurs de cette carte.

Notre projet peut être amélioré en augmentant la taille des blocs IFFT/FFT, la taille également des constellations utilisées et en modélisant notre canal par un modèle plus réaliste, car les ressources de notre carte le permettent du fait que l'occupation de cette carte par notre système n'a pas dépassé la barre des 25%. Ce travail peut être implémenté sur des processeurs DSP afin d'évaluer les paramètres réels de ce système.

References

K. Fazel et S. Kaiser. Multi-Carrier and Spread Spectrum Systems, pages 24-30.John Wiley & Sons,2003.

Quinn Lui. Design and Implementation of a Fast Fourier Transform Processor (FFT/IFFT) on an FPGA . Electronic and Communication Engineering of Curtin University of Technology. November 2006.

Jyh-Horng Wen , Gwo-Ruey Lee , Jia-Wei Liu , Te-Lung Kung. Frame synchronization, channel estimation scheme and signal compensation using. regression method in OFDM systems by .Department of Electrical Engineering, Tunghai University No. 181, Section 3, Taichung Harbor Road, Taichung, Taiwan, ROC. January 2008.

Mohammad Torabi. Adaptive modulation for space–frequency block coded OFDM systems. Département de génie électrique,école Polytechnique de Montréal, Montréal, Québec, Canada. July 2007.

S. Driz and M. Bouziani. Inter-Carrier Interference cancellation for OFDM systems. Telecommunication and Digital Signal Processing Laboratory. University of Djillali Liabes, BP 89 Sidi Bel-Abbes, Algeria. 2006.

Pierre GRUYER et Simon PAILLARD. Modélisation d'un modulateur et démodulateur OFDM. décembre 2005.

Ambarish Mukund Sule. 3 Dimensional Integrated Circuit Technology. Computer Engineering, Raleigh, North Carolina. 2007.

Magnus Nilsson. FFT, REALIZATION AND IMPLEMENTATION IN FPGA. Griffith University/Ericsson Microwave System AB 2000/2001 .

Volnei A. Pedroni. Circuit Design with VHDL. MIT Press Cambridge, London, England.2003.

Virtex-II^{TM} V2MB1000 Development Board User's Guide Version 3.0. December 2002.

XILINX. ISE Quick Start Tutorial.

Dejan M. Dramicanin , Dejan Rakic ,Slobodan Denic , Veljko Vlahovic. FPGA-based Prototyping of IEEE 802.16d Baseband Processor. SERBIAN JOURNAL OF ELECTRICAL ENGINEERING Vol. 1, No. 3, pages125 - 136. November 2004.

Altera. A Scalable OFDMA Engine for WiMAX. Version 2.1, Application Note 412. May 2007.

Altera. An OFDM FFT Kernel for WiMAX. Version 1.0, application Note 452. February 2007.

BIT ERROR RATE GENERATOR ADDITIVE WHITE GAUSSIAN NOISE GENERATOR MSS 18221 Flower Hill Way #A • Gaithersburg, Maryland 20879 U.S.A.Issued 10/24/2003.

MULTIPATH SIMULATOR. MSS 18221, Flower Hill Way. Gaithersburg, Maryland 20879 U.S.A. Juin2003.

ALTERA. Channel Estimation & Equalization for WiMAX, Version 1.1, Application Note 434.May 2007.

ALTERA. Constellation Mapper and Constellation DeMapper, version 1.1, application Note 439.May 2007.

XILINX. OPB Delta-Sigma DAC (v1.01a). Decembre 2005.

Prototyping Quadrature Amplitude Modulation for Two-way Communication on CATV Networks.

XILINX. Virtex Analog to Digital Converter . Version 1.1, application Note 440. Septembre 1999.

BPSK/QPSK/OQPSK DEMODULATOR. MSS 18221, Flower Hill Way,Gaithersburg, Maryland 20879, U.S.A. Janvier 2006.

Mohammad Azizul Hasan. Performance Evaluation of WiMAX/IEEE 802.16 OFDM Physical Layer . Master of Science in Technology. Espagne. Juin 2007.

Claudio Sacchi, Olga Zlydareva.OBJECT-ORIENTED MODEL OF SDR LIBRARY FOR WIMAX/UMTS SYSTEM BASEBAND LEVEL. UNIVERSITY OF TRENTO, DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY. Avril 2007.

Mike Paff. 802.16-2004 OFDM Receiver Baseband PHY.Master of Science in Technology, UNIVERSITY OF TRENTO, DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY. 2006.

Paul Gigliotti. Serial-to-Parallel Converter. XILINX, XAPP194 (v.1.0). Juillet 2004.

David Fritz. Spartan 3 FPGA Tutorial. Digital Logic Design. Oklahoma State University.Juin 2005.

BUI Thi Minh Tu. Performances des modulations multiporteuses OFDM/QAM suréchantillonnées en environnement radio-mobile. Equipe Signal Images et Communications SIC. juin 2005.

Seung Young Parka, Bo Seok Seob, Chung Gu Kanga. Effects of frequency offset compensation error on channel estimation for OFDM system under mobile radio channels. School of Electrical Engineering, Korea University, Seoul, South Korea. Septembre 2000.

Jiangzhang Zhua, ShuangchunWena, Qingsong Du. Space-frequency-Doppler coded OFDM over the time-varying Doppler fading channels. College of Computer and Communication, Human University, Changsha, China. 6 Avril 2007

E. Lawrey, C. J. Kikkert. PEAK TO AVERAGE POWER RATIO REDUCTION OF OFDM SIGNALS USING PEAK REDUCTION CARRIERS. James Cook University, Douglas Campus, Townsville, Queensland 4814, Australia. 2003.

Virtex-II^{TM} V2MB1000 Development Board User's Guide.

Annexe A

Les Codes Sources VHDL du sytème implémenté

L'entité Principale

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
Library UNISIM;
use UNISIM.vcomponents.all;
----------------------------------------------------------------
entity OFDM is
port ( DATA : in std_logic_vector (8 downto 1);
clk_in : in std_logic;
aff1_data_out : out std_logic_vector(7 downto 1);
aff2_data_out : out std_logic_vector(7 downto 1));
end OFDM;
----------------------------------------------------------------
architecture Behavioral of OFDM is
signal clk :std_logic;
signal paralel1,paralel2, paralel3, paralel4 : std_logic;
signal paralel5,paralel6, paralel7, paralel8 : std_logic;
signal Imap1 : integer range -3 to 3;
signal Qmap1 : integer range -3 to 3;
signal Imap2 : integer range -3 to 3;
signal Qmap2 : integer range -3 to 3;
signal IT1, QT1,IT2, QT2,IT3,QT3,IT4,QT4 : integer range -8 to 8;
signal Icp1,Icp2,Icp3,Icp4,Icp5 : integer range -8 to 8;
signal Qcp1,Qcp2, Qcp3, Qcp4, Qcp5 :integer range -8 to 8;
signal frameQ : integer range -8 to 8;
signal frameI : integer range -8 to 8;
signal retardI0, retardI1, retardI2,retardQ0, retardQ1, retardQ2 : integer range -8 to
8;
signal recuI, recuQ : integer range -10 to 10;
signal Icp1_rec,Icp2_rec,Icp3_rec,Icp4_rec,Icp5_rec : integer range -8 to 8;
signal Qcp1_rec,Qcp2_rec, Qcp3_rec, Qcp4_rec, Qcp5_rec :integer range -8 to 8;
signal IT1_rec, QT1_rec,IT2_rec, QT2_rec,IT3_rec,QT3_rec,IT4_rec,QT4_rec : in-
teger range -8 to 8;
```

```
    signal Imap1_rec : integer range -3 to 3 ;
    signal Qmap1_rec : integer range -3 to 3 ;
    signal Imap2_rec : integer range -3 to 3 ;
    signal Qmap2_rec : integer range -3 to 3 ;
    signal data_paralel1_rec,data_paralel2_rec, data_paralel3_rec, data_paralel4_rec :
std_logic ;
    signal data_paralel5_rec,data_paralel6_rec, data_paralel7_rec, data_paralel8_rec :
std_logic ;
    signal data_out : std_logic_vector (8 downto 1) ;
    ------------------------------------------------
    component divfreq is
    Port ( clk_in : in std_logic ;
    clk_out : out std_logic) ;
    end component divfreq ;
    ------------------------------------------------------
    component SPC_emi
    port ( clk : in std_logic ;
    DATA : in std_logic_vector (8 downto 1) ;
    paralel1_out, paralel2_out,paralel3_out,paralel4_out : out std_logic ;
    paralel5_out, paralel6_out,paralel7_out,paralel8_out : out std_logic) ;
    end component SPC_emi ;
    ------------------------------------------------
    component mapper
    port ( clk :in std_logic ;
    map1_in, map2_in, map3_in, map4_in : in std_logic ;
    map5_in, map6_in, map7_in, map8_in : in std_logic ;
    I1_out, Q1_out,I2_out, Q2_out : out integer range -3 to 3) ;
    end component mapper ;
    ------------------------------------------------
    component IFFT
    port ( clk :in std_logic ;
    I1_in, Q1_in,I2_in, Q2_in : in integer range -3 to 3 ;
    IFT1_out, QFT1_out : out integer range -8 to 8 ;
    IFT2_out, QFT2_out : out integer range -8 to 8 ;
    IFT3_out, QFT3_out : out integer range -8 to 8 ;
    IFT4_out, QFT4_out : out integer range -8 to 8 ) ;
    end component IFFT ;
    ------------------------------------------------
    component CPadd
    port (clk :std_logic ;
    Qcp1_in, Icp1_in :in integer range -8 to 8 ;
    Qcp2_in, Icp2_in :in integer range -8 to 8 ;
              Qcp3_in, Icp3_in :in integer range -8 to 8 ;
              Qcp4_in, Icp4_in :in integer range -8 to 8 ;
```

```
        Qcp1_out,Qcp2_out, Qcp3_out, Qcp4_out, Qcp5_out : out integer range -8
to 8;
        Icp1_out,Icp2_out,Icp3_out,Icp4_out,Icp5_out : out integer range -8 to 8);
    end component CPadd;
    ------------------------------------------------
    component PSC_emi
        port (clk : IN std_logic;
        IPS1,IPS2,IPS3,IPS4, IPS5 :IN integer range -8 to 8;
        QPS1,QPS2,QPS3,QPS4, QPS5 :IN integer range -8 to 8;
        SerdataI_out : OUT integer range -8 to 8;
        SerdataQ_out : OUT integer range -8 to 8);
    end component PSC_emi;
    ------------------------------------------------------------------------
    component canal
    port ( clk : in std_logic;
    info_emiseI : in integer range -8 to 8;
    info_emiseQ : in integer range -8 to 8;
    null_delayI : out integer range -8 to 8;
    ein_delayI :out integer range -8 to 8;
    zwei_delayI : out integer range -8 to 8;
    null_delayQ : out integer range -8 to 8;
    ein_delayQ :out integer range -8 to 8;
    zwei_delayQ : out integer range -8 to 8);
    end component canal;
    ------------------------------------------------
    component canal_out
    port ( clk : in std_logic;
    null_delayI_supp : in integer range -8 to 8;
    ein_delayI_supp :in integer range -8 to 8;
    zwei_delayI_supp : in integer range -8 to 8;
    null_delayQ_supp : in integer range -8 to 8;
    ein_delayQ_supp :in integer range -8 to 8;
    zwei_delayQ_supp : in integer range -8 to 8;
    infoI_recue : out integer range -10 to 10;
    infoQ_recue : out integer range -10 to 10);
    end component canal_out;
    ------------------------------------------------
    component SPC_rec
    port (clk : in std_logic;
    serialI_rec_data : in integer range -10 to 10;
    serialQ_rec_data : in integer range -10 to 10;
    CPSI1r_out, CPSI2r_out,CPSI3r_out,CPSI4r_out,CPSI5r_out : out integer range -8
to 8;
```

```
    CPSQ1r_out, CPSQ2r_out,CPSQ3r_out,CPSQ4r_out,CPSQ5r_out : out integer range
-8 to 8) ;
    end component SPC_rec ;
    ------------------------------------------------
    component CPsupp
    port ( clk : IN STD_LOGIC ;
              Qcp1_rec_in,Qcp2_rec_in, Qcp3_rec_in, Qcp4_rec_in, Qcp5_rec_in : in
integer range -8 to 8 ;
              Icp1_rec_in,Icp2_rec_in,Icp3_rec_in,Icp4_rec_in,Icp5_rec_in : in integer
range -8 to 8 ;
              Qcp1_rec_out,Qcp2_rec_out, Qcp3_rec_out, Qcp4_rec_out : out integer
range -8 to 8 ;
              Icp1_rec_out,Icp2_rec_out,Icp3_rec_out,Icp4_rec_out : out integer range
-8 to 8) ;
    end component CPsupp ;
    ------------------------------------------------
    component FFT
    port ( clk :in std_logic ;
    IFT1_rec_in, IFT2_rec_in, IFT3_rec_in, IFT4_rec_in : in integer range -8 to 8 ;
    QFT1_rec_in, QFT2_rec_in, QFT3_rec_in, QFT4_rec_in : in integer range -8 to
8 ;
    I1_rec_out, Q1_rec_out,I2_rec_out, Q2_rec_out : out integer range -3 to 3) ;
    end component FFT ;
    ------------------------------------------------
    component demapper
    port ( clk :in std_logic ;
    I1rec_in, Q1rec_in,I2rec_in, Q2rec_in : in integer range -3 to 3 ;
    demap1_out, demap2_out, demap3_out, demap4_out : out std_logic ;
              demap5_out, demap6_out, demap7_out, demap8_out : out std_logic) ;
    end component demapper ;
    ------------------------------------------------
    component PSC_rec
        port (
        clk : IN std_logic ;
    PS1_rec_in ,PS2_rec_in ,PS3_rec_in ,PS4_rec_in :IN std_logic ;
              PS5_rec_in ,PS6_rec_in ,PS7_rec_in ,PS8_rec_in :IN std_logic ;
        singal_restitue : OUT std_logic_vector (8 downto 1)) ;
    end component PSC_rec ;
    ------------------------------------------------
    component decof is
    Port (clk :in std_logic ;
        aff_in : in std_logic_vector (8 downto 1) ;
        aff1_data_out : out std_logic_vector(7 downto 1) ;
        aff2_data_out : out std_logic_vector(7 downto 1)) ;
```

```
end component decof;
-------------------------------------------------------------------
begin
lab_divfreq : divfreq
    port map( clk_in => clk_in,
    clk_out=> clk );
-------------------------------------------------------------------
    lab_SPC_emi : SPC_emi
    port map(
                        clk =>clk,
                        DATA=> DATA,
                  paralel1_out => paralel1,
                        paralel2_out => paralel2,
                        paralel3_out => paralel3,
                        paralel4_out => paralel4,
                        paralel5_out => paralel5,
                        paralel6_out => paralel6,
                        paralel7_out => paralel7,
                        paralel8_out => paralel8);
-------------------------------------------------------------------
    lab_mapper : mapper
    port map (
                        clk =>clk,
map1_in => paralel1,
                        map2_in => paralel2,
                        map3_in => paralel3,
                        map4_in => paralel4,
                        map5_in => paralel5,
                        map6_in => paralel6,
                        map7_in => paralel7,
                        map8_in => paralel8,
                        I1_out=> Imap1,
                        Q1_out=> Qmap1,
                        I2_out=> Imap2,
                        Q2_out=> Qmap2 );
-------------------------------------------------------------------

    lab_IFFT : IFFT
    port map (
                        clk =>clk,
                        I1_in => Imap1,
                        Q1_in => Qmap1,
                        I2_in => Imap2,
                        Q2_in => Qmap2,
```

```
IFT1_out => IT1,
                        QFT1_out => QT1,
                        IFT2_out => IT2,
                        QFT2_out => QT2,
                        IFT3_out => IT3,
                        QFT3_out => QT3,
                        IFT4_out => IT4,
                        QFT4_out => QT4);
-------------------------------------------------------------------
    lab_CPadd :     CPadd
    port map (
                        clk =>clk,
                        Icp1_in => IT1,
Qcp1_in => QT1,
                        Icp2_in => IT2,
Qcp2_in => QT2,
                        Icp3_in => IT3,
Qcp3_in => QT3,
                        Icp4_in => IT4,
Qcp4_in => QT4,
          Qcp1_out => Qcp1,
                        Qcp2_out => Qcp2,
                        Qcp3_out => Qcp3,
                        Qcp4_out => Qcp4,
                        Qcp5_out => Qcp5,
          Icp1_out => Icp1,
                        Icp2_out => Icp2,
                        Icp3_out => Icp3,
                        Icp4_out => Icp4,
                        Icp5_out => Icp5 );
-------------------------------------------------------------------
    lab_PSC_emi : PSC_emi
    port map(
                        clk =>clk,
                        IPS1 => Icp1,
                        IPS2 => Icp2,
                        IPS3 => Icp3,
                        IPS4 => Icp4,
                        IPS5 => Icp5,
    QPS1 => Qcp1,
                        QPS2 => Qcp2,
                        QPS3 => Qcp3,
                        QPS4 => Qcp4,
                        QPS5 => Qcp5,
```

```
    SerdataI_out => frameI,
    SerdataQ_out => frameQ) ;
-------------------------------------------------------------
    lab_canal : canal
    port map(
                        clk =>clk,
info_emiseI => frameI,
info_emiseQ => frameQ,
null_delayI => retardI0,
ein_delayI => retardI1,
zwei_delayI => retardI2,
null_delayQ => retardQ0,
ein_delayQ => retardQ1,
zwei_delayQ => retardQ2) ;
-------------------------------------------------------------
    lab_canal_out : canal_out
    port map(
                        clk =>clk,
                        null_delayI_supp => retardI0,
ein_delayI_supp => retardI1,
zwei_delayI_supp => retardI2,
null_delayQ_supp => retardQ0,
ein_delayQ_supp => retardQ1,
zwei_delayQ_supp => retardQ2,
infoI_recue => recuI,
infoQ_recue => recuQ) ;
-------------------------------------------------------------
    lab_SPC_rec :      SPC_rec
    port map ( clk => clk,
                        serialI_rec_data => recuI,
                        serialQ_rec_data => recuQ,
CPSI1r_out => Icp1_rec,
                        CPSI2r_out => Icp2_rec,
                        CPSI3r_out => Icp3_rec,
                        CPSI4r_out => Icp4_rec,
                        CPSI5r_out => Icp5_rec,
CPSQ1r_out => Qcp1_rec,
                        CPSQ2r_out => Qcp2_rec,
                        CPSQ3r_out => Qcp3_rec,
                        CPSQ4r_out => Qcp4_rec,
                        CPSQ5r_out => Qcp5_rec) ;
-------------------------------------------------------------
    lab_CPsupp : CPsupp
    port map(
```

```
                    clk => clk,
                    Icp1_rec_in => Icp1_rec,
                    Icp2_rec_in => Icp2_rec,
                    Icp3_rec_in => Icp3_rec,
                    Icp4_rec_in => Icp4_rec,
                    Icp5_rec_in => Icp5_rec,
                    Qcp1_rec_in => Qcp1_rec,
                    Qcp2_rec_in => Qcp2_rec,
                    Qcp3_rec_in => Qcp3_rec,
                    Qcp4_rec_in => Qcp4_rec,
                    Qcp5_rec_in => Qcp5_rec,
                    Icp1_rec_out => IT1_rec,
                    Icp2_rec_out => IT2_rec,
                    Icp3_rec_out => IT3_rec,
                    Icp4_rec_out => IT4_rec,
        Qcp1_rec_out => QT1_rec,
                    Qcp2_rec_out => QT2_rec,
                    Qcp3_rec_out => QT3_rec,
                    Qcp4_rec_out => QT4_rec );
```

```
    lab_FFT : FFT
    port map (
                    clk => clk,
IFT1_rec_in => IT1_rec,
                    IFT2_rec_in => IT2_rec,
                    IFT3_rec_in => IT3_rec,
                    IFT4_rec_in => IT4_rec,
QFT1_rec_in => QT1_rec,
                    QFT2_rec_in => QT2_rec,
                    QFT3_rec_in => QT3_rec,
                    QFT4_rec_in => QT4_rec,
I1_rec_out => Imap1_rec,
                    Q1_rec_out => Qmap1_rec,
                    I2_rec_out => Imap2_rec,
                    Q2_rec_out => Qmap2_rec);
```

```
    lab_demapper : demapper
    port map (
                    clk => clk,
I1rec_in => Imap1_rec,
                    Q1rec_in => Qmap1_rec,
                    I2rec_in => Imap2_rec,
                    Q2rec_in => Qmap2_rec,
demap1_out => data_paralel1_rec,
```

```
                    demap2_out => data_paralel2_rec,
                    demap3_out => data_paralel3_rec,
                    demap4_out => data_paralel4_rec,
                    demap5_out => data_paralel5_rec,
                    demap6_out => data_paralel6_rec,
                    demap7_out => data_paralel7_rec,
                    demap8_out => data_paralel8_rec);
-------------------------------------------------------
    lab_PSC_rec  : PSC_rec
    port map (
                    clk => clk,
                    PS1_rec_in => data_paralel1_rec,
                    PS2_rec_in => data_paralel2_rec,
                    PS3_rec_in => data_paralel3_rec,
                    PS4_rec_in => data_paralel4_rec,
                    PS5_rec_in => data_paralel5_rec,
                    PS6_rec_in => data_paralel6_rec,
                    PS7_rec_in => data_paralel7_rec,
                    PS8_rec_in => data_paralel8_rec,
    singal_restitue=> data_out);
-------------------------------------------------------
lab_decof  : decof
    port map (
                    clk=> clk,
    aff_in => data_out,
                    aff1_data_out=>aff1_data_out,
                    aff2_data_out => aff2_data_out);
-------------------------------------------------------
end Behavioral;
```

1. L'emetteur

   –
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity divfreq is
Port ( clk_in : in std_logic;
clk_out : out std_logic);
end divfreq;
architecture Behavioral of divfreq is
signal som, cs : integer range 0 to 99999999;
begin
n1 : process(cs)
begin
if cs >= 99999999 then
```

```
clk_out <= '1';
else
clk_out <= '0';
end if;
end process n1;
n2 : process (clk_in, som)
begin
if (clk_in'EVENT AND clk_in='1') then
cs <= som;
end if;
end process n2;
n3 : process (cs)
begin
if (cs = 99999999) then
som <= 0;
else
som <= cs + 1;
end if;
end process n3;
end Behavioral;
```

–
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
Library UNISIM;
use UNISIM.vcomponents.all;
entity SPC_emi is
port ( clk : in std_logic;
DATA : in std_logic_vector (8 downto 1);
paralel1_out, paralel2_out,paralel3_out,paralel4_out : out std_logic;
paralel5_out, paralel6_out,paralel7_out,paralel8_out : out std_logic);
end SPC_emi;
architecture Behavioral of SPC_emi is
signal amina : integer range 0 to 100;
signal akli1,akli2,akli3,akli4,akli5,akli6,akli7,akli8 : std_logic;
begin
process (clk,amina)
BEGIN
IF (clk'EVENT AND clk='1') THEN
akli1 <= DATA(1);
akli2 <= DATA(2);
akli3 <= DATA(3);
akli4 <= DATA(4);
akli5 <= DATA(5);
```

```
akli6 <= DATA(6);
akli7 <= DATA(7);
akli8 <= DATA(8);
IF (amina=2) THEN
paralel1_out <= akli1;
paralel2_out <= akli2;
paralel3_out <= akli3;
paralel4_out <= akli4;
end if;
if (amina=3) then
paralel5_out <= akli5;
paralel6_out <= akli6;
paralel7_out <= akli7;
paralel8_out <= akli8;
end if;
amina <= amina+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mapper is
port (clk :in std_logic;map1_in, map2_in, map3_in, map4_in : in std_logic;
map5_in, map6_in, map7_in, map8_in : in std_logic;
I1_out, Q1_out,I2_out, Q2_out : out integer range -3 to 3);
end mapper;
architecture Behavioral of mapper is
signal debut_mapping : integer range 0 to 100;
begin
process(clk, map1_in, map2_in, map3_in, map4_in,debut_mapping )
begin
IF (clk'EVENT AND clk='1') THEN
if (debut_mapping=4) then
case (map1_in) is
when '0' =>
case (map2_in) is
when '0' =>
case (map3_in) is
when '0' =>
case (map4_in) is
when '0' => I1_out <= -3;Q1_out <= 3;
when OTHERS => I1_out <= -3;Q1_out <= -3;
```

```
end case;
when OTHERS =>
case (map4_in) is
when '0' => I1_out <= -3; Q1_out <= 1;
when OTHERS => I1_out <= -3; Q1_out <= -1;
end case;
end case;
when OTHERS =>
case (map3_in) is
when '0' =>
case (map4_in) is
when '0' => I1_out <= 3; Q1_out <= 3;
                        when OTHERS => I1_out <= 3; Q1_out <= -3;
end case;
when OTHERS =>
case (map4_in) is
when '0' => I1_out <= 3; Q1_out <= 1;
when OTHERS => I1_out <= 3; Q1_out <= -1;
end case;
end case;
end case;
when OTHERS =>
case (map2_in) is
when '0' =>
case (map3_in) is
when '0' =>
case (map4_in) is
when '0' => I1_out <= -1; Q1_out <= 3;
when OTHERS => I1_out <= -1; Q1_out <= -3;
end case;
when OTHERS =>
case (map4_in) is
when '0' => I1_out <= -1; Q1_out <= 1;
when OTHERS => I1_out <= -1; Q1_out <= -1;
end case;
end case;
when OTHERS =>
case (map3_in) is
when '0' =>
case (map4_in) is
when '0' => I1_out <= 1; Q1_out <= 3;
when OTHERS => I1_out <= 1; Q1_out <= -3;
end case;
when OTHERS =>
```

```
case (map4_in) is
when '0' => I1_out <= 1 ; Q1_out <= 1 ;
when OTHERS => I1_out <= 1 ; Q1_out <= -1 ;
end case ;
end case ;
end case ;
END case ;
end if ;
end if ;
end process ;
process(clk, map5_in, map6_in, map7_in, map8_in,debut_mapping )
begin
IF (clk'EVENT AND clk='1') THEN
if (debut_mapping=5) then
case (map5_in) is
when '0' =>
case (map6_in) is
when '0' =>
case (map7_in) is
when '0' =>
case (map8_in) is
when '0' => I2_out <= -3 ;Q2_out <= 3 ;
when OTHERS => I2_out <= -3 ;Q2_out <= -3 ;
end case ;
when OTHERS =>
case (map8_in) is
when '0' => I2_out <= -3 ; Q2_out <= 1 ;
when OTHERS => I2_out <= -3 ; Q2_out <= -1 ;
end case ;
end case ;
when OTHERS =>
case (map7_in) is
when '0' =>
case (map8_in) is
when '0' => I2_out <= 3 ; Q2_out <= 3 ;
                        when OTHERS => I2_out <= 3 ; Q2_out <= -3 ;
end case ;
when OTHERS =>
case (map8_in) is
when '0' => I2_out <= 3 ; Q2_out <= 1 ;
when OTHERS => I2_out <= 3 ; Q2_out <= -1 ;
end case ;
end case ;
end case ;
```

```
when OTHERS =>
case (map6_in) is
when '0' =>
case (map7_in) is
when '0' =>
case (map8_in) is
when '0' => I2_out <= -1; Q2_out <= 3;
when OTHERS => I2_out <= -1; Q2_out <= -3;
end case;
when OTHERS =>
case (map8_in) is
when '0' => I2_out <= -1; Q2_out <= 1;
when OTHERS => I2_out <= -1; Q2_out <= -1;
end case;
end case;
when OTHERS =>
case (map7_in) is
when '0' =>
case (map8_in) is
when '0' => I2_out <= 1; Q2_out <= 3;
when OTHERS => I2_out <= 1; Q2_out <= -3;
end case;
when OTHERS =>
case (map8_in) is
when '0' => I2_out <= 1; Q2_out <= 1;
when OTHERS => I2_out <= 1; Q2_out <= -1;
end case;
end case;
end case;
END case;
end if;
debut_mapping <= debut_mapping+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity IFFT is
port ( clk :in std_logic;
I1_in, Q1_in,I2_in, Q2_in : in integer range -3 to 3;
IFT1_out, QFT1_out : out integer range -8 to 8;
IFT2_out, QFT2_out : out integer range -8 to 8;
```

```
IFT3_out, QFT3_out : out integer range -8 to 8 ;
IFT4_out, QFT4_out : out integer range -8 to 8 ) ;
end IFFT ;
architecture Behavioral of IFFT is
signal start_IFFT : integer range 0 to 100 ;
begin
PROCESS(clk, start_IFFT)
begin
IF (clk'EVENT AND clk='1') THEN
if (start_IFFT=6) then
IFT1_out <= I1_in + I2_in +2 ;
IFT2_out <= I1_in - I2_in +2 ;
IFT3_out <= I1_in + I2_in -2 ;
IFT4_out <= I1_in - I2_in -2 ;
QFT1_out <= Q1_in + Q2_in +2 ;
QFT2_out <= Q1_in - Q2_in -2 ;
QFT3_out <= Q1_in + Q2_in -2 ;
QFT4_out <= Q1_in - Q2_in +2 ;
end if ;
start_IFFT <= start_IFFT+1 ;
end if ;
end process ;
end Behavioral ;
– library IEEE ;
use IEEE.STD_LOGIC_1164.ALL ;
use IEEE.STD_LOGIC_ARITH.ALL ;
use IEEE.STD_LOGIC_UNSIGNED.ALL ;
entity CPadd is
port ( clk :std_logic ;
Qcp1_in, Icp1_in :in integer range -8 to 8 ;
Qcp2_in, Icp2_in :in integer range -8 to 8 ;
          Qcp3_in, Icp3_in :in integer range -8 to 8 ;
          Qcp4_in, Icp4_in :in integer range -8 to 8 ;
          Qcp1_out,Qcp2_out, Qcp3_out, Qcp4_out, Qcp5_out : out integer range
-8 to 8 ;
          Icp1_out,Icp2_out,Icp3_out,Icp4_out,Icp5_out : out integer range -8
to 8) ;
end CPadd ;
architecture Behavioral of CPadd is
signal start_CPadd : integer range 0 to 100 ;
begin
PROCESS(clk, start_CPadd)
begin
IF (clk'EVENT AND clk='1') THEN
```

```
if (start_CPadd=7) then
Icp1_out <= Icp4_in;
Qcp1_out <= Qcp4_in;
Icp2_out <= Icp1_in;
Qcp2_out <= Qcp1_in;
Icp3_out <= Icp2_in;
Qcp3_out <= Qcp2_in;
Icp4_out <= Icp3_in;
Qcp4_out <= Qcp3_in;
Icp5_out <= Icp4_in;
Qcp5_out <= Qcp4_in;
end if;
start_CPadd<=start_CPadd+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PSC_emi is
port (clk : IN std_logic;
      IPS1,IPS2,IPS3,IPS4, IPS5 :IN integer range -8 to 8;
      QPS1,QPS2,QPS3,QPS4, QPS5 :IN integer range -8 to 8;
      SerdataI_out : OUT integer range -8 to 8;
      SerdataQ_out : OUT integer range -8 to 8);
end PSC_emi;
architecture Behavioral of PSC_emi is
signal start_PSC_emi : integer range 0 to 100;
signal selec_PSC_emi : integer range 0 to 1000;
begin
process(clk, selec_PSC_emi,start_PSC_emi )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_PSC_emi=9 OR start_PSC_emi=10 OR start_PSC_emi=11 OR start_PSC_emi=
OR start_PSC_emi=13 ORstart_PSC_emi=14) then
case (selec_PSC_emi) is
when 9 => SerdataI_out <= IPS1;
when 10 => SerdataI_out <= IPS2;
when 11 => SerdataI_out <= IPS3;
when 12 => SerdataI_out <= IPS4;
when 13 => SerdataI_out <= IPS5;
when 14 => SerdataI_out <= IPS5;
when others =>SerdataI_out <= 0;
```

```
end case;
end if;
end if;
end process;
process(clk, selec_PSC_emi,start_PSC_emi )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_PSC_emi=9 OR start_PSC_emi=10 OR start_PSC_emi=11 OR start_PSC_emi=
OR start_PSC_emi=13 ORstart_PSC_emi=14) then
case (selec_PSC_emi) is
when 9 => SerdataQ_out <= QPS1;
when 10 => SerdataQ_out <= QPS2;
when 11 => SerdataQ_out <= QPS3;
when 12 => SerdataQ_out <= QPS4;
when 13 => SerdataQ_out <= QPS5;
when 14 => SerdataQ_out <= QPS5;
when others =>SerdataQ_out <= 0;
end case;
end if;
selec_PSC_emi<=selec_PSC_emi+1;
start_PSC_emi<=start_PSC_emi+1;
end if;
end process;
end Behavioral;
```

2. Le Canal
   -
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity canal is
port ( clk : in std_logic;
info_emiseI : in integer range -8 to 8;
info_emiseQ : in integer range -8 to 8;
null_delayI : out integer range -8 to 8;
ein_delayI :out integer range -8 to 8;
zwei_delayI : out integer range -8 to 8;
null_delayQ : out integer range -8 to 8;
ein_delayQ :out integer range -8 to 8;
zwei_delayQ : out integer range -8 to 8);
end canal;
architecture Behavioral of canal is
signal start_canal : integer range 0 to 100;
signal selec_canal : integer range 0 to 1000;
signal massiI1,massiI2,massiI3,massiI4,massiI5 : integer range -8 to 8;
```

```
signal massiI6,massiI7,massiI8,massiI9,massiI10 : integer range -8 to 8;
signal massiI11,massiI12,massiI13,massiI14,massiI15 : integer range -8 to 8;
signal massiQ1,massiQ2,massiQ3,massiQ4,massiQ5 : integer range -8 to 8;
signal massiQ6,massiQ7,massiQ8,massiQ9,massiQ10 : integer range -8 to 8;
signal massiQ11,massiQ12,massiQ13,massiQ14,massiQ15 : integer range -8 to 8;
begin
process(clk, selec_canal,start_canal )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_canal=10 OR start_canal=11 OR start_canal=12 OR start_canal=13
OR start_canal=14 OR start_canal=15) then
case (selec_canal) is
when 10 => massiI1 <= info_emiseI;massiI6 <= info_emiseI;massiI11 <= info_emiseI;
when 11 => massiI2 <= info_emiseI;massiI7 <= info_emiseI;massiI12 <= info_emiseI;
when 12 => massiI3 <= info_emiseI;massiI8 <= info_emiseI;massiI13 <= info_emiseI;
when 13 => massiI4 <= info_emiseI;massiI9 <= info_emiseI;massiI14 <= info_emiseI;
when 14 => massiI5 <= info_emiseI;massiI10 <= info_emiseI;massiI15 <=
info_emiseI;
when 15 => massiI5 <= info_emiseI;massiI10 <= info_emiseI;massiI15 <=
info_emiseI;
when others =>massiI1 <= info_emiseI;massiI10 <= info_emiseI;massiI15 <=
info_emiseI;end case;
end if;
if (start_canal=11 OR start_canal=12 OR start_canal=13 OR start_canal=14
OR start_canal=15 OR start_canal=16) then
case (selec_canal) is
when 11 => null_delayI <= massiI1;
when 12 => null_delayI <= massiI2;
when 13 => null_delayI <= massiI3;
when 14 => null_delayI <= massiI4;
when 15 => null_delayI <= massiI5;
when 16 => null_delayI <= massiI5;
when others => null_delayI <= massiI5;
end case;
end if;
if (start_canal=12 OR start_canal=13 OR start_canal=14 OR start_canal=15
OR start_canal=16 OR start_canal=17) then
case (selec_canal) is
when 12 => ein_delayI <= massiI6;
when 13 => ein_delayI <= massiI7;
when 14 => ein_delayI <= massiI8;
when 15 => ein_delayI <= massiI9;
when 16 => ein_delayI <= massiI10;
when 17 => ein_delayI <= massiI10;
```

```
when others => ein_delayI <= massiI10;
end case;
end if;
if (start_canal=13 OR start_canal=14 OR start_canal=15 OR start_canal=16
OR start_canal=17 OR start_canal=18) then
case (selec_canal) is
when 13 => zwei_delayI <= massiI11;
when 14 => zwei_delayI <= massiI12;
when 15 => zwei_delayI <= massiI13;
when 16 => zwei_delayI <= massiI14;
when 17 => zwei_delayI <= massiI15;
when 18 => zwei_delayI <= massiI15;
when others => zwei_delayI <= massiI15;
end case;
end if;
end if;
end process;
process(clk, selec_canal,start_canal )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_canal=10 OR start_canal=11 OR start_canal=12 OR start_canal=13
OR start_canal=14 OR start_canal=15) then
case (selec_canal) is
when 10 => massiQ1 <= info_emiseQ;massiQ6 <= info_emiseQ;massiQ11 <=
info_emiseQ;
when 11 => massiQ2 <= info_emiseQ;massiQ7 <= info_emiseQ;massiQ12 <=
info_emiseQ;
when 12 => massiQ3 <= info_emiseQ;massiQ8 <= info_emiseQ;massiQ13 <=
info_emiseQ;
when 13 => massiQ4 <= info_emiseQ;massiQ9 <= info_emiseQ;massiQ14 <=
info_emiseQ;
when 14 => massiQ5 <= info_emiseQ;massiQ10 <= info_emiseQ;massiQ15 <=
info_emiseQ;
when 15 => massiQ5 <= info_emiseQ;massiQ10 <= info_emiseQ;massiQ15 <=
info_emiseQ;
when others =>massiQ1 <= info_emiseQ;massiQ10 <= info_emiseQ;massiQ15
<= info_emiseQ;
end case;
end if;
if (start_canal=11 OR start_canal=12 OR start_canal=13 OR start_canal=14
OR start_canal=15 OR start_canal=16) then
case (selec_canal) is
when 11 => null_delayQ <= massiQ1;
when 12 => null_delayQ <= massiQ2;
```

```
when 13 => null_delayQ <= massiQ3;
when 14 => null_delayQ <= massiQ4;
when 15 => null_delayQ <= massiQ5;
when 16 => null_delayQ <= massiQ5;
when others => null_delayQ <= massiQ5;
end case;
end if;
if (start_canal=12 OR start_canal=13 OR start_canal=14 OR start_canal=15
OR start_canal=16 OR start_canal=17) then
case (selec_canal) is
when 12 => ein_delayQ <= massiQ6;
when 13 => ein_delayQ <= massiQ7;
when 14 => ein_delayQ <= massiQ8;
when 15 => ein_delayQ <= massiQ9;
when 16 => ein_delayQ <= massiQ10;
when 17 => ein_delayQ <= massiQ10;
when others => ein_delayQ <= massiQ10;
end case;
end if;
if (start_canal=13 OR start_canal=14 OR start_canal=15 OR start_canal=16
OR start_canal=17 OR start_canal=18) then
case (selec_canal) is
when 13 => zwei_delayQ <= massiQ11;
when 14 => zwei_delayQ <= massiQ12;
when 15 => zwei_delayQ <= massiQ13;
when 16 => zwei_delayQ <= massiQ14;
when 17 => zwei_delayQ <= massiQ15;
when 18 => zwei_delayQ <= massiQ15;
when others => zwei_delayQ <= massiQ15;
end case;
end if;
selec_canal<=selec_canal+1;
start_canal<=start_canal+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity canal_out is
port ( clk : in std_logic;
null_delayI_supp : in integer range -8 to 8;
ein_delayI_supp :in integer range -8 to 8;
```

```
zwei_delayI_supp : in integer range -8 to 8;
null_delayQ_supp : in integer range -8 to 8;
ein_delayQ_supp :in integer range -8 to 8;
zwei_delayQ_supp : in integer range -8 to 8;
infoI_recue : out integer range -10 to 10;
infoQ_recue : out integer range -10 to 10);
end canal_out;
architecture Behavioral of canal_out is
signal start_canal_out : integer range 0 to 100;
signal selec_canal_out : integer range 0 to 1000;
signal massiI1,massiI2,massiI3,massiI4,massiI5 : integer range -8 to 8;
signal massiI6,massiI7,massiI8,massiI9,massiI10 : integer range -8 to 8;
signal massiI11,massiI12,massiI13,massiI14,massiI15 : integer range -8 to 8;
signal massiQ1,massiQ2,massiQ3,massiQ4,massiQ5 : integer range -8 to 8;
signal massiQ6,massiQ7,massiQ8,massiQ9,massiQ10 : integer range -8 to 8;
signal massiQ11,massiQ12,massiQ13,massiQ14,massiQ15 : integer range -8 to 8;
begin
process(clk, selec_canal_out,start_canal_out )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_canal_out=12 OR start_canal_out=13 OR start_canal_out=14 OR
start_canal_out=15 OR start_canal_out=16 ORstart_canal_out=17) then
case (selec_canal_out) is
when 12 => massiI1 <= null_delayI_supp;
when 13 => massiI2 <= null_delayI_supp;
when 14 => massiI3<= null_delayI_supp;
when 15 => massiI4 <=null_delayI_supp;
when 16 => massiI5 <=null_delayI_supp;when 17 => massiI5<= null_delayI_supp;
when others => massiI5 <= null_delayI_supp;
end case;
end if;
if (start_canal_out=13 OR start_canal_out=14 OR start_canal_out=15 OR
start_canal_out=16 OR start_canal_out=17 OR start_canal_out=18) then
case (selec_canal_out) is
when 13 => massiI6 <=ein_delayI_supp;
when 14 => massiI7 <=ein_delayI_supp;
when 15 => massiI8 <= ein_delayI_supp;
when 16 => massiI9 <= ein_delayI_supp;
when 17 => massiI10 <= ein_delayI_supp;
when 18 => massiI10 <=ein_delayI_supp;
when others => massiI10 <= ein_delayI_supp;
end case;
end if;
if (start_canal_out=14 OR start_canal_out=15 OR start_canal_out=16 OR
```

```
start_canal_out=17 OR start_canal_out=18 OR start_canal_out=19) then
case (selec_canal_out) is
when 14 => massiI11<= zwei_delayI_supp;
when 15 => massiI12 <=zwei_delayI_supp;
when 16 => massiI13 <= zwei_delayI_supp;
when 17 => massiI14 <= zwei_delayI_supp;
when 18 => massiI15 <=zwei_delayI_supp;
when 19 => massiI15 <= zwei_delayI_supp;
when others => massiI15 <= zwei_delayI_supp;
end case;
end if;
if (start_canal_out=15 OR start_canal_out=16 OR start_canal_out=17 OR
start_canal_out=18 OR start_canal_out=19 OR start_canal_out=20) then
case (selec_canal_out) is
when 15 => infoI_recue<= massiI1+massiI6-massiI11+2;
when 16 => infoI_recue <=massiI2+massiI7-massiI12;
when 17 => infoI_recue <= massiI3+massiI8-massiI13;
when 18 => infoI_recue <= massiI4+massiI9-massiI14;
when 19 => infoI_recue <=massiI5+massiI10-massiI15;
when 20 => infoI_recue <= massiI5+massiI10-massiI15;
when others => infoI_recue <= massiI5+massiI10-massiI15;
end case;
end if;
end if;
end process;
process(clk, selec_canal_out,start_canal_out )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_canal_out=12 OR start_canal_out=13 OR start_canal_out=14 OR
start_canal_out=15 OR start_canal_out=16 OR start_canal_out=17) then
case (selec_canal_out) is
when 12 => massiQ1 <= null_delayQ_supp;
when 13 => massiQ2 <= null_delayQ_supp;
when 14 => massiQ3<= null_delayQ_supp;
when 15 => massiQ4 <=null_delayQ_supp;
when 16 => massiQ5 <=null_delayQ_supp;
when 17 => massiQ5<= null_delayQ_supp;
when others => massiQ5 <= null_delayQ_supp;
end case;
end if;
if (start_canal_out=13 OR start_canal_out=14 OR start_canal_out=15 OR
start_canal_out=16 OR start_canal_out=17 OR start_canal_out=18) then
case (selec_canal_out) is
when 13 => massiQ6 <=ein_delayQ_supp;
```

```
when 14 => massiQ7 <=ein_delayQ_supp;
when 15 => massiQ8 <= ein_delayQ_supp;
when 16 => massiQ9 <= ein_delayQ_supp;
when 17 => massiQ10 <= ein_delayQ_supp;
when 18 => massiQ10 <=ein_delayQ_supp;
when others => massiQ10 <= ein_delayQ_supp;
end case;
end if;
if (start_canal_out=14 OR start_canal_out=15 OR start_canal_out=16 OR
start_canal_out=17 OR start_canal_out=18 OR start_canal_out=19) then
case (selec_canal_out) is
when 14 => massiQ11<= zwei_delayQ_supp;
when 15 => massiQ12 <=zwei_delayQ_supp;
when 16 => massiQ13 <= zwei_delayQ_supp;
when 17 => massiQ14 <= zwei_delayQ_supp;
when 18 => massiQ15 <=zwei_delayQ_supp;
when 19 => massiQ15 <= zwei_delayQ_supp;
when others => massiQ15 <= zwei_delayQ_supp;
end case;
end if;
if (start_canal_out=15 OR start_canal_out=16 OR start_canal_out=17 OR
start_canal_out=18 OR start_canal_out=19 OR start_canal_out=20) then
case (selec_canal_out) is
when 15 => infoQ_recue<= massiQ1+massiQ6-massiQ11+2;
when 16 => infoQ_recue <=massiQ2+massiQ7-massiQ12;
when 17 => infoQ_recue <= massiQ3+massiQ8-massiQ13;
when 18 => infoQ_recue <= massiQ4+massiQ9-massiQ14;
when 19 => infoQ_recue <=massiQ5+massiQ10-massiQ15;
when 20 => infoQ_recue <= massiQ5+massiQ10-massiQ15;
when others => infoQ_recue <= massiQ5+massiQ10-massiQ15;
end case;
end if;
selec_canal_out <= selec_canal_out+1;
start_canal_out <= start_canal_out+1;
end if;
end process;
end Behavioral;
```

3. Le Récepteur
   -
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity SPC_rec is
port ( clk : in std_logic;
```

```
serialI_rec_data : in integer range -8 to 8;
serialQ_rec_data : in integer range -8 to 8;
CPSI1r_out, CPSI2r_out,CPSI3r_out,CPSI4r_out,CPSI5r_out : out integer range
-8 to 8;
CPSQ1r_out, CPSQ2r_out,CPSQ3r_out,CPSQ4r_out,CPSQ5r_out : out inte-
ger range -8 to 8);
end SPC_rec;
architecture Behavioral of SPC_rec is
signal start_SPC_rec : integer range 0 to 100;
signal papa : integer range 0 to 100 :=0;
signal signI1,signI2,signI3,signI4,signI5 : integer range -8 to 8;
signal signQ1,signQ2, signQ3, signQ4, signQ5 : integer range -8 to 8;
begin
process (clk,papa,start_SPC_rec )
BEGIN
IF (clk'EVENT AND clk='1') THEN
if (start_SPC_rec=16 OR start_SPC_rec=17 OR start_SPC_rec=18 OR start_SPC_rec=1
OR start_SPC_rec=20 OR start_SPC_rec=21) then
case (papa) is
when 16 => signI1 <= serialI_rec_data; signQ1 <= serialQ_rec_data;
when 17 => signI2 <= serialI_rec_data; signQ2 <= serialQ_rec_data;
when 18 => signI3 <= serialI_rec_data; signQ3 <= serialQ_rec_data;
when 19 => signI4 <= serialI_rec_data; signQ4 <= serialQ_rec_data;
when 20 => signI5 <= serialI_rec_data; signQ5 <= serialQ_rec_data;
when 21 => signI5 <= serialI_rec_data; signQ5 <= serialQ_rec_data;
when others => signI5 <= serialI_rec_data; signQ5 <= serialQ_rec_data;
end case;
end if;
IF (papa=21) THEN
CPSI1r_out <= signI1;
CPSI2r_out <= signI2;
CPSI3r_out <= signI3;
CPSI4r_out <= signI4;
CPSI5r_out <= signI5;
CPSQ1r_out <= signQ1;
CPSQ2r_out <= signQ2;
CPSQ3r_out <= signQ3;
CPSQ4r_out <= signQ4;
CPSQ5r_out <= signQ5;
end if;
papa <= papa+1;
start_SPC_rec <= start_SPC_rec+1;
end if;
end process;
```

```
end Behavioral;
– Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity CPsupp is
port (clk : IN STD_LOGIC;
        Qcp1_rec_in,Qcp2_rec_in, Qcp3_rec_in, Qcp4_rec_in, Qcp5_rec_in :
in integer range -8 to 8;
        Icp1_rec_in,Icp2_rec_in,Icp3_rec_in,Icp4_rec_in,Icp5_rec_in : in in-
teger range -8 to 8;
        Qcp1_rec_out,Qcp2_rec_out, Qcp3_rec_out, Qcp4_rec_out : out in-
teger range -8 to 8;
        Icp1_rec_out,Icp2_rec_out,Icp3_rec_out,Icp4_rec_out : out integer range
-8 to 8);
end CPsupp;
architecture Behavioral of CPsupp is
signal start_CPsupp : integer range 0 to 100;
begin
process (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
if (start_CPsupp=22) then
Icp1_rec_out <= Icp2_rec_in;
Icp2_rec_out <= Icp3_rec_in;
Icp3_rec_out <= Icp4_rec_in;
Icp4_rec_out <= Icp5_rec_in+Icp1_rec_in-Icp1_rec_in;
Qcp1_rec_out <= Qcp2_rec_in;
Qcp2_rec_out <= Qcp3_rec_in;
Qcp3_rec_out <= Qcp4_rec_in;
Qcp4_rec_out <= Qcp5_rec_in+Qcp1_rec_in-Qcp1_rec_in;
end if;
start_CPsupp <= start_CPsupp+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FFT is
port (clk :in std_logic;
IFT1_rec_in, IFT2_rec_in, IFT3_rec_in, IFT4_rec_in : in integer range -8 to
8;
```

```
QFT1_rec_in, QFT2_rec_in, QFT3_rec_in, QFT4_rec_in : in integer range -8
to 8;
I1_rec_out, Q1_rec_out,I2_rec_out, Q2_rec_out : out integer range -3 to 3);
end FFT;
architecture Behavioral of FFT is
signal start_FFT : integer range 0 to 100;
BEGIN
PROCESS(clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
if (start_FFT=23) then
I1_rec_out <= (IFT1_rec_in+IFT2_rec_in+IFT3_rec_in+IFT4_rec_in)/4;
I2_rec_out <= (IFT1_rec_in-IFT2_rec_in+IFT3_rec_in-IFT4_rec_in)/4;
Q1_rec_out <= (QFT1_rec_in+QFT2_rec_in+QFT3_rec_in+QFT4_rec_in)/4;
Q2_rec_out <= (QFT1_rec_in-QFT2_rec_in+QFT3_rec_in-QFT4_rec_in)/4;
end if;
start_FFT <= start_FFT+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity demapper is
port(clk :in std_logic;
I1rec_in, Q1rec_in,I2rec_in, Q2rec_in : in integer range -3 to 3;
demap1_out, demap2_out, demap3_out, demap4_out : out std_logic;
demp5_out, demap6_out, demap7_out, demap8_out : out std_logic);
end demapper ;architecture Behavioral of demapper is
signal start_demapper :integer range 0 to 100;
begin
process (clk,I1rec_in, Q1rec_in,start_demapper)
BEGIN
IF (clk'EVENT AND clk='1') THEN
if (start_demapper=24) then
case (I1rec_in) is
when -3 =>
    case (Q1rec_in) is
when -3 => demap1_out <= '0'; demap2_out <= '0';demap3_out <= '0'; de-
map4_out <= '1';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= '0'; demap2_out <='0';demap3_out <='1'; de-
```

```
map4_out <= '1';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= '0'; demap2_out <= '0';demap3_out <= '1'; de-
map4_out <= '0';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <= '0'; demap2_out <= '0';demap3_out <= '0'; de-
map4_out <= '0';
    end case;
    when -2 =>
    case (Q1rec_in) is
when -3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    end case;
    when -1 =>
    case (Q1rec_in) is
when -3 => demap1_out <= '1'; demap2_out <= '0';demap3_out <= '0'; de-
map4_out <= '1';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= '1'; demap2_out <= '0';demap3_out <= '1'; de-
map4_out <= '1';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= '1'; demap2_out <= '0';demap3_out <= '1'; de-
map4_out <= '0';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <=
'Z';demap4_out <= 'Z';
when 3 => demap1_out <= '1'; demap2_out <= '0';demap3_out <= '0'; de-
map4_out <= '0';
    end case;
```

```
    when 0 =>
    case (Q1rec_in) is
when -3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    end case;
when 1 =>
    case (Q1rec_in) is
when -3 => demap1_out <= '1'; demap2_out <='1';demap3_out <= '0'; de-
map4_out <= '1';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= '1'; demap2_out <= '1';demap3_out <= '1'; de-
map4_out <= '1';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= '1'; demap2_out <= '1';demap3_out <= '1'; de-
map4_out <= '0';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <='1'; demap2_out <= '1';demap3_out <= '0'; de-
map4_out <= '0';
    end case;
    when 2 =>
    case (Q1rec_in) is
when -3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
```

```
when 1 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z'; de-
map4_out <= 'Z';
    end case;
    when 3 =>
    case (Q1rec_in) is
when -3 => demap1_out <= '0'; demap2_out <= '1';demap3_out <= '0'; de-
map4_out <= '1';
    when -2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when -1 => demap1_out <= '0'; demap2_out <= '1';demap3_out <= '1'; de-
map4_out <= '1';
    when 0 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 1 => demap1_out <= '0'; demap2_out <= '1';demap3_out <= '1'; de-
map4_out <='0';
    when 2 => demap1_out <= 'Z'; demap2_out <= 'Z';demap3_out <= 'Z';
demap4_out <= 'Z';
when 3 => demap1_out <= '0'; demap2_out <= '1';demap3_out <= '0'; de-
map4_out <='0';
    end case;
end case;
end if;
end if;
end process;
process (clk,I2rec_in, Q2rec_in,start_demapper)
BEGIN
IF (clk'EVENT AND clk='1') THEN
if (start_demapper=24) then
case (I2rec_in) is
when -3 =>
    case (Q2rec_in) is
when -3 => demap5_out <= '0'; demap6_out <= '0';demap7_out <= '0'; de-
map8_out <= '1';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= '0'; demap6_out <='0';demap7_out <='1'; de-
map8_out <= 1';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out = 'Z';
when 1 => demap5_out <= '0'; demap6_out <= '0';demap7_out <= '1'; de-
```

```
map8_out <= '0'
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <=
'Z'; demap8_out <= 'Z'; when 3 => demap5_out <= '0'; demap6_out <=
'0';demap7_out <= '0'; demap8_out <= '0';
    end case;
    when -2 =>
    case (Q2rec_in) is
when -3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
map8_out <= 'Z';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
map8_out <= 'Z';
    end case;
    when -1 =>
    case (Q2rec_in) is
when -3 => demap5_out <= '1'; demap6_out <= '0';demap7_out <= '0'; de-
map8_out <= '1';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= '1'; demap6_out <= '0';demap7_out <= '1'; de-
map8_out <= '1';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= '1'; demap6_out <= '0';demap7_out <= '1'; de-
map8_out <= '0';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <=
'Z';demap8_out <= 'Z';
when 3 => demap5_out <= '1'; demap6_out <= '0';demap7_out <= '0'; de-
map8_out <= '0';
    end case;
    when 0 =>
    case (Q2rec_in) is
when -3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
```

```
demap8_out <= 'Z';
when -1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
map8_out <= 'Z';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
map8_out <= 'Z';
    end case;
when 1 =>
    case (Q2rec_in) is
when -3 => demap5_out <= '1'; demap6_out <='1';demap7_out <= '0'; de-
map8_out <= '1';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= '1'; demap6_out <= '1';demap7_out <= '1'; de-
map8_out <= '1';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= '1'; demap6_out <= '1';demap7_out <= '1'; de-
map8_out <= '0';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 3 => demap5_out <='1'; demap6_out <= '1';demap7_out <= '0'; de-
map8_out <= '0';
    end case;
    when 2 =>
    case (Q2rec_in) is
when -3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
map8_out <= 'Z';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 3 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z'; de-
```

```
map8_out <= 'Z';
    end case;
    when 3 =>
    case (Q2rec_in) is
when -3 => demap5_out <= '0'; demap6_out <= '1';demap7_out <= '0'; de-
map8_out <= '1';
    when -2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when -1 => demap5_out <= '0'; demap6_out <= '1';demap7_out <= '1'; de-
map8_out <= '1';
    when 0 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 1 => demap5_out <= '0'; demap6_out <= '1';demap7_out <= '1'; de-
map8_out <='0';
    when 2 => demap5_out <= 'Z'; demap6_out <= 'Z';demap7_out <= 'Z';
demap8_out <= 'Z';
when 3 => demap5_out <= '0'; demap6_out <= '1';demap7_out <= '0'; de-
map8_out <='0';
    end case;
end case;
end if;
start_demapper<=start_demapper+1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PSC_rec is
port (clk : IN std_logic;
PS1_rec_in ,PS2_rec_in ,PS3_rec_in ,PS4_rec_in :IN std_logic;
        PS5_rec_in ,PS6_rec_in ,PS7_rec_in ,PS8_rec_in :IN std_logic;
    singal_restitue : OUT std_logic_vector (8 downto 1));
end PSC_rec;
architecture Behavioral of PSC_rec is
signal start_PSC_rec :integer range 0 to 100;
begin
process(clk, start_PSC_rec )
begin
IF (clk'EVENT AND clk='1') THEN
if (start_PSC_rec=25) then
singal_restitue(1) <= PS1_rec_in;
singal_restitue(2) <= PS2_rec_in;
```

```
singal_restitue(3) <= PS3_rec_in;
singal_restitue(4) <= PS4_rec_in;
singal_restitue(5) <= PS5_rec_in;
singal_restitue(6) <= PS6_rec_in;
singal_restitue(7) <= PS7_rec_in;
singal_restitue(8) <= PS8_rec_in;
end if;
start_PSC_rec <= start_PSC_rec + 1;
end if;
end process;
end Behavioral;
– library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity decof is
Port (clk :in std_logic;
      aff_in : in std_logic_vector (8 downto 1);
      aff1_data_out : out std_logic_vector(7 downto 1);
      aff2_data_out : out std_logic_vector(7 downto 1)); – MSB
end decof;
architecture Behavioral of decof is
signal inter : std_logic_vector (8 downto 1);
signal af1 : std_logic_vector (4 downto 1);
signal af2 : std_logic_vector (4 downto 1);
begin
inter(1) <= aff_in(1);
inter(2) <= aff_in(2);
inter(3) <= aff_in(3);
inter(4) <= aff_in(4);
af1(1) <= inter(1);
af1(2) <= inter(2);
af1(3) <= inter(3);
af1(4) <= inter(4);
process(clk,af1)
begin
IF (clk'EVENT AND clk='1') THEN
case (af1) is
when "0000" => aff1_data_out <= "1111110";
when "1000" => aff1_data_out<="1111111";
when "0100" => aff1_data_out<="0110011";
when "1100" => aff1_data_out<= "1001110";
when "0010" => aff1_data_out<= "1101101";
when "1010" => aff1_data_out<= "1110111";
```

```
when "0110" => aff1_data_out<= "1011111";
when "1110" => aff1_data_out<= "1001111";
when "0001" => aff1_data_out<="0110000";
when "1001" => aff1_data_out<= "1111011";
when "0101" => aff1_data_out<="1011011";
when "1101" => aff1_data_out<= "0111101";
when "0011" => aff1_data_out<="1111001";
when "1011" => aff1_data_out<= "0011111";
when "0111" =>aff1_data_out<= "1110000";
when "1111" => aff1_data_out<= "1000111";
when others => aff1_data_out<= "0000000";
END case;
end if;
end process;
inter(5) <= aff_in(5);
inter(6) <= aff_in(6);
inter(7) <= aff_in(7);
inter(8) <= aff_in(8);
af2(1) <= inter(5);
af2(2) <= inter(6);
af2(3) <= inter(7);
af2(4) <= inter(8);
process(clk,af2)
begin
IF (clk'EVENT AND clk='1') THEN
case (af2) is
when "0000" => aff2_data_out <= "1111110";
when "1000" => aff2_data_out<="1111111";
when "0100" => aff2_data_out<="0110011";
when "1100" => aff2_data_out<= "1001110";
when "0010" => aff2_data_out<= "1101101";
when "1010" => aff2_data_out<= "1110111";
when "0110" => aff2_data_out<= "1011111";
when "1110" => aff2_data_out<= "1001111";
when "0001" => aff2_data_out<="0110000";
when "1001" => aff2_data_out<= "1111011";
when "0101" => aff2_data_out<="1011011";
when "1101" => aff2_data_out<= "0111101";
when "0011" => aff2_data_out<="1111001";
when "1011" => aff2_data_out<= "0011111";
when "0111" => aff2_data_out<= "1110000";
when "1111" => aff2_data_out<= "1000111";
when others => aff2_data_out<= "0000000";
END case;
```

```
end if;
end process;
end Behavioral;
```