5/02

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

ECOLE NATIONALE POLYTECHNIQUE
DEPARTEMENT D'ELECTRONIQUE

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Mémoire de fin d'études
En vue de l'obtention du diplôme d'ingénieur d'état

THEME

Implémentation d'une FFT basée sur l'algorithme CORDIC sur le circuit FPGA

Etudié et soutenu par :

- M. BOUNOUA Amine
- Mlle BOUSBIA Hind

Devant le jury composé de :

Madame Guerti	Présidente
Mr Boussekssou	Examinateur
Mr Bousbia S	Promoteur
Mr Bélouchrani	Co-promoteur

Promotion 2002

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Je dédie ce travail à :

Mes très chers parents.

Mes très chers frères et sœurs :Amel, Chafika, Feriel, Nassim et Tarik.

Mon beau frère Omar.

Mon adorable petit neveux Chakib.

Mes tantes et oncles.

Ma chere binome.

Mrs bousbia salah et kheir-eddine .

Et a tous mes amis

AMINE.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Je dédie ce travail:

A mes très chers parents.

A mes très chers frères et sœurs: Myriam, Raouf, Kenzi, Amel, Samiha, Halim et Mohamed.

A mon oncle Kheir-Eddine.

A mes chers grands-parents.

A mes chers oncles et tantes.

A mon binôme et sa famille.

A mes amies: Yasmina, Sonia et Lilia.

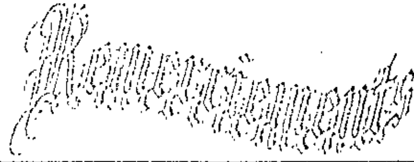
A tous mes amis de Skikda.

A tous mes amis d'Alger.

A tous mes amis de Constantine.

A Aziz et Camila.

Hind



المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Que Monsieur Bousbia-Salah et Monsieur Bélouchrani, nos encadreurs trouvent ici l'expression de notre profonde gratitude pour nous avoir proposé un sujet de recherche intéressant, actuel et plein d'avenir. Qu'ils sachent que leurs conseils avisés, leurs soutiens et leurs encouragements nous ont été très précieux.

Nous tenons également à exprimer toute notre reconnaissance à :

- *Monsieur Saadoun pour tous ces conseils et l'attention qu'il nous a accordé.*
- *Madame M.Guerti qui nous a fait l'honneur de présider notre jury.*
- *Monsieur B.Bouseksou qui nous a fait l'honneur de nous examiner.*
- *Monsieur Bousbia Kheir-Eddine pour toute l'aide qu'il nous a apporté.*

Nos collègues de laboratoire Monsieur Dedjel et Mademoiselle Djeghlaf pour le soutien et l'aide qu'ils nous ont apporté.

Notre vive reconnaissance va également à tous nos professeurs qui n'ont ménagé aucun efforts pour nous donner un enseignement de qualité.

A tous nos camarades de classe avec qui ont a passé de difficiles et d'agréables moments.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

ملخص

هذا العمل يتعلق دراسة غرس خوارزمية تحويلة فوريي السريعة FFT بالكورديك على دارة مبرمجة من نوع FPGA،تحقيق هذا العمل على هذه الدارة جديد، خاصة وسائل التفسير مثل VHDL ، الذي اردنا احسانه و ايضا كل وسائل التطوير: التمثيل،مخطط المركبات و الة الحالات.

مفاتيح: FFT CORDIC FPGA VHDL استخلاص

Résumé

Ce projet est consacré à l'étude et à l'implémentation de l'algorithme de la transformée de fourier rapide(FFT) basée sur l'algorithme CORDIC sur circuits programmables de type (FPGA), la réalisation de projets sur ce type de circuits étant très récente , notamment des outils de description comme le VHDL que nous tenions à maîtriser ainsi que tous les autres outils de développement :simulateur ,schématique ou machine d'état. La partie simulation temporelle a aussi été aborder .

Mots cles: CORDIC, FFT,FPGA, VHDL, synthèse.

Abstract

In this project, the design of a CORDIC algorithm based FFT processor is presented. The different units of the processor are implemented in Fields Programmable Gate Array .
We lead down a complete development of our application using the fundamental tools of XILINX:VHDL editor, SHEMATIC editor, FSM editor , timing simulations have been done also.

Keywords: FPGA; CORDIC; FFT; VHDL; Synthesis.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

SOMMAIRE

I - LA TRANSFORMÉE DE FOURIER DISCRÈTE 5
I.1 Introduction 5
I.2 Définition de la TFD 6
I.3 L'algorithme du CORDIC 10
I.3.1 Introduction 10
I.3.2 Description de l'algorithme 10
I.4 Conclusion 13
II - LE VHDL & LA TECHNOLOGIE FPGA 15
II.1 Le VHDL 15
II.1.1 Introduction 15
II.1.2 Principes généraux 16
II.1.2.a Description descendante : le « top down design » 16
II.1.2.b Les niveaux d'abstraction 17
II.1.2.c Simulation et/ou synthèse 18
II.1.2.d L'extérieur de la boîte noire : une « ENTITE » 20
II.1.2.e Le fonctionnement interne : une « ARCHITECTURE » 21
II.2 La technologie FPGA 25
II.2.1 Introduction 25
II.2.2 Les LCA de XILINX 25
II.2.3 Les cellules logiques de base des LCA 29
II.2.4 Les différents types d'interconnexions 31
II.3 Convention de XILINX SUR LES FPGA 35
II.4 Conclusion 36
III - CONCEPTION D'ARCHITECTURE ET SIMULATION FONCTIONNELLE 38
III.1 Introduction 38
III.2 L'unité de génération d'adresses 39
III.2.1 Architecture fonctionnelle 41
III.2.2 Simulation fonctionnelle et interprétation des résultats 45
III.3 L'unité papillon (Butterfly) 46
III.3.1 L'unité SCALE 47
III.3.1.a Architecture fonctionnelle 48
III.3.1.b Simulation fonctionnelle et interprétation des résultats 49
III.3.2 L'unité CORDIC 51
III.3.2.a Résolution d'opérations mathématiques par le CORDIC 51
III.3.2.b Simulations et résultats 52
III.3.2.c Architecture fonctionnelle : 57
III.4 L'unité de contrôle 59
III.5 - Conclusion 61
IV - SYNTHÈSE, PALCEMENT-ROUTAGE ET SIMULATION TEMPORELLE 63
IV.1 Synthèse et placement-routage 63
IV.1.1 Introduction 63
IV.1.2 Synthèse : le concept 63
IV.1.3 Critères de performances 64
IV.1.3.a Puissance de calcul 64
IV.1.4 Etapes de la synthèse 67
IV.1.5 Synthèse et placement-routage du bloc de génération d'adresses 68
IV.1.6 La synthèse du bloc SCALE 69
La synthèse du bloc CORDIC 72

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

IV.2 SIMULATION TEMPORELLE 73
IV.2.1 Introduction 73
IV.2.2 Simulation temporelle du bloc de génération d'adresse : 73
IV.2.3 Simulation temporelle du bloc SCALE 74
IV.2.4 Simulation temporelle du bloc CORDIC 75
IV.3 CONCLUSION 77
V - CONCLUSION GÉNÉRALE 78

ANNEXES

Annexe -A- : *Le code source VHDL du bloc de génération d'adresses*
Annexe -B- : *Le code source VHDL du bloc SCALE*
Annexe -C- : *Le code source VHDL du bloc CORDIC*
Annexe -D- : *Le code source VHDL du bloc contrôleur*
Annexe -E- : *Les différents rapports issus de l'implémentation*

REFFERENCES BIBLIOGRAPHIQUES

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Liste des figures

Figure 1. Spectre d'une suite d'impulsions. ---------- 6
Figure 2. Le calcule de la FFT en utilisant l'entrelacement fréquentiel pour N = 8. ---------- 9
Figure 3. L'opération du papillon---------- 9
Figure 4. Rotation d'un vecteur V d'un angle α. ---------- 11
Figure 5. Diagramme en Y introduit par Gajski décrivant les trois vues d'une architecture---------- 17
Figure 6. mise en évidence du gaspillage de ressources dans un PLD classique.---------- 25
Figure 7. la structure adoptée par XILINX pour ces LCA. ---------- 27
Figure 8. structure détaillée d'un IOB. ---------- 29
Figure 9. Structure détaillée d'un CLB. ---------- 31
Figure 10. Mise en évidence de quelques PIP---------- 32
Figure 11. Emplacements et possibilités des matrices de commutation---------- 33
Figure 12. Utilisation des possibilités d'interconnexions directes. ---------- 33
Figure 13. *emplacement des longues lignes.* ---------- 35
Figure 14. Synoptique du schéma général ---------- 39
Figure 15. L'organigramme de l'unité de génération d'adresses---------- 40
Figure 16. Le bloc de génération d'adresses.---------- 45
Figure 17. Simulation de l'unité de génération d'adresse.---------- 45
Figure 18. L'unité Papillon. ---------- 47
Figure 19. Le bloc SCALE. ---------- 49
Figure 20. Simulation fonctionnelle du bloc SCALE. ---------- 49
Figure 21 .Schema fonctionnel du cordic. ---------- 52
Figure 22. Format de la donnée dans le mode rotation . ---------- 52
Figure 23. Chronogramme du CORDIC dans le mode rotation---------- 53
Figure 24. Format des données X etY dans le mode vecteur.---------- 55
Figure 25. Format de la donnée E . ---------- 55
Figure 26. *Chronogramme du CORDIC dans le mode vecteur* ---------- 56
Figure 27. Le maintient de l'angle dans le 1er et 4ème quadrant---------- 57
Figure 28. Le bloc CORDIC. ---------- 58
Figure 29.Chronogramme de simulation du contrôleur. ---------- 60
Figure 30. *Schéma global du processeur FFT*---------- 61
Figure 31. Modèle de calcul de la fréquence maximum. ---------- 66
Figure 32. Le bloc de génération d'adresses après la phase de placement et de routage.---------- 68
Figure 33. Circuit représentant le bloc de génération d'adresses . ---------- 69
Figure 34. Le bloc SCALE après la phase de placement et routage---------- 70
Figure 35. Circuit représentant le bloc SCALE ---------- 71
Figure 36. Le bloc CORDIC après la phase placement et routage---------- 72
Figure 37. Circuit représentant le bloc CORDIC ---------- 72
Figure 38. Chronogramme de la simulation temporelle du bloc de génération d'adresses à f=32MHz---------- 74
Figure 39. Chronogramme de la simulation temporelle du bloc de génération d'adresses à f=35MHz---------- 74
Figure 40.Chronogramme de la simulation temporelle du bloc SCALE à f=38MHz ---------- 75
Figure 41 Chronogramme de la simulation temporelle du bloc SCALE à f=50MHz ---------- 75
Figure 42 Chronogramme de la simulation temporelle du bloc CORDIC à f=20MHz ---------- 76
Figure 43 Chronogramme de la simulation temporelle du bloc CORDIC à f=25MHz---------- 76

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

INTRODUCTION GENERALLE

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Introduction

Les besoins en information sont énorme et notamment en biomedical, l'électronique permet d'accéder rapidement à ces sources d'information utiles et valides.

Le signal est la transcription et l'enregistrement de phénomènes physiques, mais d'une partie seulement de ces phénomènes : un ECG ne représente pas toute l'activité électrique du cœur, mais seulement les variations de potentiel en un nombre limité de points cutanés, une coupe échographique ne montre qu'une partie du cœur, une sonde intra cardiaque peut enregistrer bruits et pressions ou l'un seulement de ces deux phénomènes dans une gamme limitée de fréquences, ...etc. Ce que nous détectons n'est qu'une partie du phénomène physiologique, limitée par les capacités des détecteurs. Un phénomène en trois dimensions (volume anatomique) ou même en quatre dimensions (volume + temps) est traduit sur un document en deux dimensions, film, papier, écran d'ordinateur. Un phénomène périodique n'est recueilli que dans une gamme de fréquences ou une durée de temps limitée. *Le signal est donc obligatoirement réducteur d'une réalité* et non pas la réalité elle-même.

Dans le traitement des signaux biomédicaux, l'analyse spectrale joue un rôle clé, particulièrement dans la présentation des données acquises et les mettre dans un format clinique utile pour faciliter le diagnostique établi par les médecins praticiens.

Parmi les analyseurs des spectres existants, on trouve ceux qui sont essentiellement basés sur l'emploi de la transformée de Fourier discrète (TFD) par les techniques numériques de transformation de Fourier rapide (FFT, Fast Fourier Tranformer).

Il devient plus aisé de passer par la décomposition en fréquences de base (spectre de Fourier) des variations du signal de l'image en totalité.

Cette méthode est déjà utilisée dans la construction de l'image scanographique ou IRM ; le filtrage ou la compression d'image relève alors des mêmes méthodes mathématiques. Les variations d'amplitude du signal sur une ligne d'image représentent un signal qui sera décomposé en diverses fréquences dont chacune a elle-même une amplitude, ce qui constitue un *spectre de fréquences*. À ce stade, on peut manipuler chacune des fréquences, amplifier les fréquences basses et réduire les fréquences élevées, ce qui représente un filtre passe-bas ou inversement.

Favoriser les fréquences moyennes en éliminant les fréquences extrêmes peut mettre en évidence des éléments d'image d'une dimension donnée, par exemple des nodules pulmonaires mesurant entre 5 et 10 mm.

Le bruit *(parasite)* altère le signal ; si l'on connaît la fréquence d'un facteur de bruit, il suffit d'éliminer cette fréquence pour faire disparaître cet élément de bruit ; or un certain nombre de bruits ont une cause connue, un amplificateur a une fréquence de résonance propre, les 50 périodes d'alimentation, la fréquence de rotation d'un enregistreur magnétique peuvent être isolées. On connaît actuellement en musique les enregistrements anciens convertis sur disque compact ; la qualité semble extraordinaire, car les bruits parasites du disque 78 tours ont été éliminés de cette manière.

Les méthodes numériques de filtrage sont indispensables dans ce cas et devront être choisies avec soin. Elles reposent toutes sur le principe, sur un échantillon de courte durée, d'analyse spectrale obtenue par le biais d'une conversion temps - fréquence suivie d'une élimination des indésirables et d'une restauration ultérieure du signal analogique débarrassé du bruit.
Toutes les procédures employées dérivent de la même constatation due à Fourier, à savoir qu'un signal périodique peut-être décomposé aisément en ses composantes individuelles dont on sait qu'elles comportent un terme continu, un terme sinusoïdal dit fondamental et un nombre infini d'harmoniques dont les amplitudes vont décroissant quand l'ordre de l'harmonique augmente, certains termes pouvant d'ailleurs être nuls. L'extrapolation de cette décomposition à un échantillon fini dans le temps, d'une part, et la mise au point de circuits électroniques spécialisés capables d'effectuer les calculs en temps quasi réel a permis un grand développement de cette technique dite de transformée rapide de Fourier (FFT).

Nous avons choisi dans notre travail de concevoir une FFT basée sur l'algorithme CORDIC qui peut être utilisée pour traiter des signaux biomédicaux et d'implémenter sur circuits FPGA les trois modules essentiels la constituant et qui sont :

- Module de génération d'adresses.
- Module de l'algorithme CORDIC.
- Module de normalisation (SCALE).

L'implémentation s'est faite a l'aide de l'outils de développement XILINX FOUNDATION v1.5. contenant tous les outils de simulation et de synthèse.

Le plan de la thèse :

Dans l'introduction générale nous avons donné un brève aperçu sur les signaux biomédicaux et l'importance de la FFT dans leur analyse et nous avons décrit les motivations pour la réalisation de cette dernière.

Dans le premier chapitre nous allons donné les bases théoriques de la DFT, de l'algorithme FFT qui fait appel à l'algorithme CORDIC pour le calcul des SINUS et des COSINUS.

Enchaînons ensuite avec un deuxième chapitre dans lequel nous allons aborder un langage de description de circuits électroniques qui est le VHDL, nécessaire à l'implémentation de nos algorithmes sur circuits FPGA.

Le troisième chapitre aborde la technologie des circuits FPGA qui serons les circuits cibles de notre application.

Dans le quatrièmes chapitre nous allons passer à la conception de notre architecture en présentant les différents modules la constituant ainsi que leurs simulations fonctionnelles.

Le cinquième chapitre sera consacré à la synthèse des différents blocs qui constituant notre architecture.

Le dernier chapitre présente le comportement temporel des différents circuits issus de la synthèse.

Finalement une conclusion qui résume les résultats obtenus et propose quelques recommandations pour les travaux futurs.

Chapitre 1

La Transformée de Fourier Discrète

I.1- INTRODUCTION 5

I.2- DEFINITION DE LA TFD 6

I.3- L'ALGORITHME CORDIC 10

I.3-1- INTRODUCTION 10

I.3-2- DESCRIPTION DE L'ALGORITHME 10

I.4- CONCLUSION 13

I - La transformée de Fourier discrète

I.1 Introduction [1, 3]

La transformée de Fourier Discrète s'introduit quand il s'agit de calculer la transformée de Fourier d'une fonction à l'aide d'un calculateur numérique. En effet un tel opérateur ne peut traiter que des nombres et de plus en quantité limitée par la taille de sa mémoire. Il s'en suit que la transformée de Fourier :

$$S(f) = \int_{-\infty}^{+\infty} S(t).e^{-j2\pi ft}\,dt$$

doit être adaptée, d'une part en remplaçant le signal S(t) par des nombres S(nT) qui représentent un échantillonnage de ce signal et d'autre part en limitant l'ensemble des nombres sur lesquels portent les calculs à une valeur finie N. Le calcul fournit alors des nombres $S^{*}(f)$ définis par : $S^{*}(f) = \sum_{n=0}^{N-1} S(nT)e^{-j2\pi ft}\,dt$

Comme le calculateur a une puissance de calcul limitée, il ne peut fournir ces résultats que pour un nombre fini de valeurs de la fréquence f, qu'il est naturel de choisir multiples d'un certain pas de fréquence Δf, alors:

$$S^{*}(k\Delta f) = \sum_{n=0}^{N-1} S(nT).e^{-j2\pi nk\Delta fT}$$

Un choix simplificateur intéressant consiste à prendre : $\Delta f = \dfrac{1}{NT}$.

D'autre part, la transformée ainsi calculée se présente sous la forme de valeurs discrètes comme l'indique l'exemple suivant:

Calculons le spectre d'une fonction périodique Sp(t) de période T. Une telle fonction peut être considérée comme résultante du produit de convolution de la fonction S(t) qui prend les valeurs de Sp(t) sur une période et s'annule en dehors et de la distribution ponctuelle U(t). Il en résulte la relation suivante entre les transformées de Fourier:

$$Sp(f) = U(f).S(f) = \frac{1}{T}.\sum_{n=-\infty}^{+\infty} S(\frac{n}{T}).\delta(f-\frac{n}{T})$$

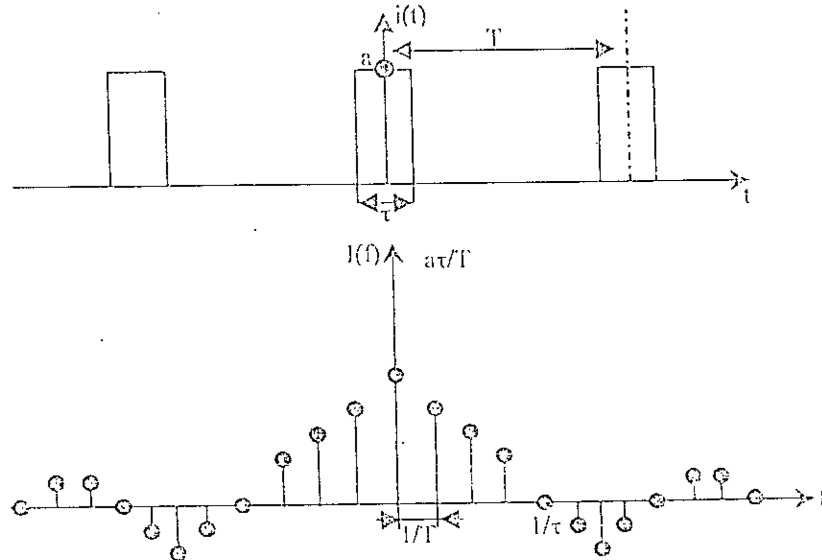


Figure 1. Spectre d'une suite d'impulsions.

Il apparaît que le spectre de la fonction périodique Sp(t) est un spectre de raies qui constituent un échantillonnage du spectre de la fonction prise sur une période. L'échantillonnage dans l'espace des fréquences correspond à une périodicité dans l'espace des temps. Cette interprétation est utile dans l'analyse numérique des spectres.[1,2,3].

I.2 Définition de la TFD[1, 2, 3, 4]

La transformée de Fourier discrète (DFT) pour un nombre N d'échantillons complexes est:

$$F(r) = \sum_{k=0}^{N-1} f(k)W^{r.k}, r = 0,1,...,N-1 \quad ....(1)$$

Sachant que : $W = e^{(-\frac{2\pi j}{N})}$.

La FFT est la méthode la plus adéquate et la plus efficace pour le calcul de la DFT d'un nombre N d'échantillons d'une donnée discrète avec une complexité temporelle égale à $(N\log_2 N)$ à l'opposition de la valeur (N^2) en employant la méthode directe.

La première étape de l'algorithme de la FFT consiste à séparer la donnée de l'entrée $f(k)$ en nombres pairs et impairs $e(k)$ et $d(k)$ respectivement comme suit:

$$e(k) = f(2k). \quad ....(2)$$

$$d(k) = f(2k+1), \quad \text{où } k = 0,1,\ldots,\frac{N}{2}-1$$

A présent l'équation (1) peut être écrite sous la forme suivante:

$$F(r) = \sum_{k=0}^{\frac{N}{2}-1}\left\{e(k)W^{2.r.k} + d(k)W^{2r.k+r}\right\} \quad ....(3)$$

$$F(r) = E(r) + W^r.D(r) \quad ....(4)$$

$$\text{où} \quad E(r) = \sum_{k=0}^{\frac{N}{2}-1} e(k)W^{2rk}, \quad r = 0,1,\ldots,\frac{N}{2}-1. \quad ....(5)$$

$$\text{et} \quad D(r) = \sum_{k=0}^{\frac{N}{2}-1} d(k).W^{2rk}, \quad r = 0,1,\ldots,\frac{N}{2}-1.$$

$E(r)$ et $D(r)$ représentent les DFT de $e(k)$ et $d(k)$ qui s'étendent sur $\frac{N}{2}$ points

Dans l'algorithme de la FFT nous utilisons:

L'entrelacement temporel où la suite d'éléments $f(k)$ peut être décomposée en deux suites entrelacées, celle des éléments d'indice pair et celle des éléments d'indice impair.

La propriété de symétrie où la suite d'éléments $f(k)$ est divisée en deux parties égales $g(k)$ contenant les N/2 premiers échantillons et $h(k)$ contenant les N/2 derniers échantillons comme suit:

$$g(k) = f(k)$$

$$\text{et} \quad ....(6)$$

$$h(k) = f(k+\frac{N}{2}), \qquad k = 0,1,\ldots,\frac{N}{2}-1$$

l'équation (1) sera donc de la forme:

$$F(k) = \sum_{k=0}^{\frac{N}{2}-1} \left\{ g(k).W^{rk} + h(k).W^{r.k+r.\frac{N}{2}} \right\} \quad \text{...............(7)}$$

Si $B(r)$ et $C(r)$ sont les transformées des deux suites paire et impaire nous aurons:

$$B(r) = F(2r)$$

et

$$C(r) = F(2r+1), \qquad r = 0,1,\ldots,\frac{N}{2}-1.$$

donc

$$B(r) = \sum_{k=0}^{\frac{N}{2}-1} [g(k) + h(k)] W^{2.r.k}$$

et

$$C(r) = \sum_{k=0}^{\frac{N}{2}-1} [g(k) - h(k)] W^{k}.W^{2rk}$$

$B(r)$et $C(r)$ représentent les DFT des fonctions $[g(k) + h(k)]$ et $[g(k) - h(k)]W^{k}$ sur N/2 éléments respectivement.

Donc, en employant l'entrelacement temporel et la symétrie DFT sur une longueur de N éléments peut être divisée en deux DFT sur une longueur de N/2 points chacune et ainsi de suite.

Ceci est illustré à la figure (2), comme exemple nous avons pris N = 8.

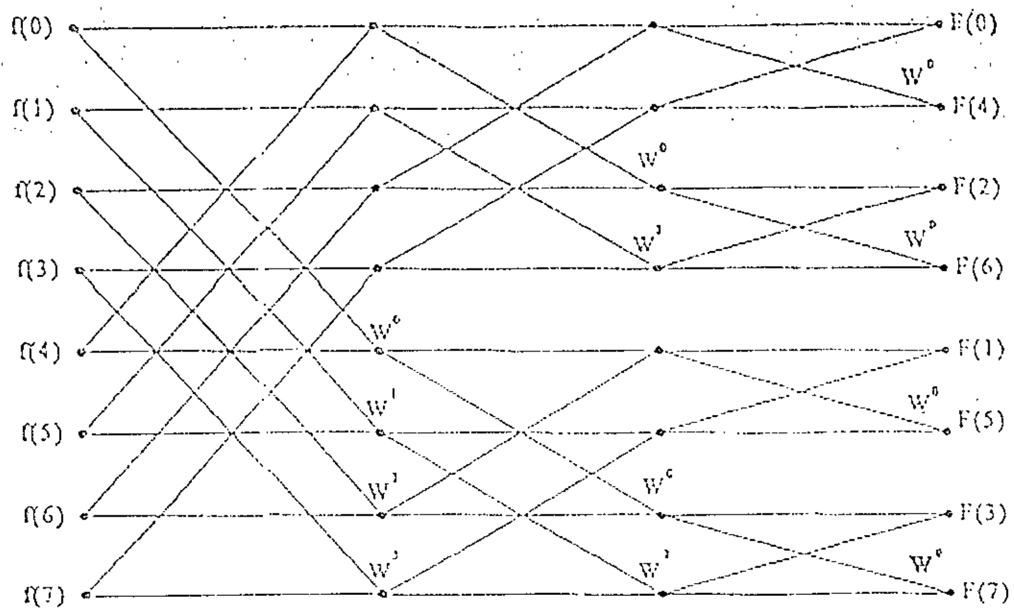


Figure 2. Le calcule de la FFT en utilisant la symétrie pour N = 8.

L'opération élémentaire de la figure (2) est le papillon ou le croisillon représenté à la figure 3 et ayant la forme suivante:

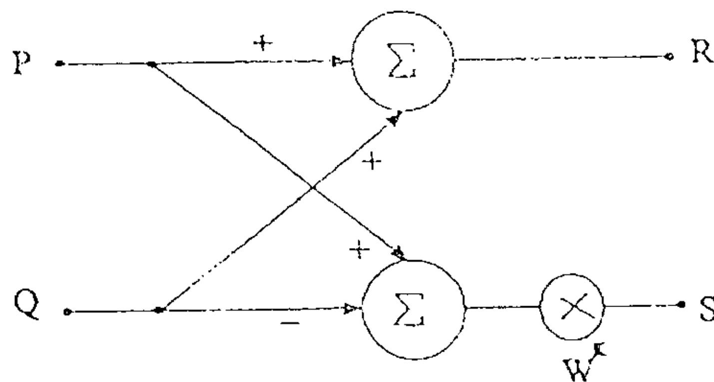


Figure 3. L'opération du papillon

$R = P + Q$

$S = (P - Q).W^{k}$ ....(10)

En écrivant l'équation (10) sous sa forme développée en termes réels et imaginaires nous obtenons :

$$
\begin{aligned}
R_{re} &= P_{re} + Q_{re} \\
R_{im} &= P_{im} + Q_{im} \qquad \text{.............(11)} \\
S_{re} &= (P_{re} - Q_{re}).Cos(k\theta) + (P_{im} - Q_{im}).Sin(k\theta). \\
S_{im} &= -(P_{re} - Q_{re}).Sin(k\theta) + (P_{im} - Q_{im}).Cos(k\theta).
\end{aligned}
$$

Les deux derniers termes de la série d'équations (11) représentent essentiellement une opération de rotation plane qui peut être efficacement calculée en appliquant l'algorithme CORDIC que nous allons aborder dans le paragraphe suivant.

I.3 L'algorithme du CORDIC[13]

I.3.1 Introduction

L'algorithme CORDIC (COordinate Rotation Digital Computing) a été introduit par Jack E.Volder, en 1959. Cet algorithme est basé sur un processus itératif pouvant nous fournir des résultats suivant deux modes, le mode rotation et le mode vectoriel.

Cet algorithme représente une méthode simple et efficace qui permet le calcul d'une gamme de fonctions complexes. S'appuyant sur une technique de décalage-addition et de rotation vectorielle, l'algorithme calcule par approximation la plupart des fonctions basées sur la trigonométrie. Beaucoup de calculatrices commerciales utilisent cet algorithme pour résoudre les fonctions sinus, cosinus, arctangente et racine carrée. Comme la plupart des algorithmes performants, l'efficacité du CORDIC repose sur sa simplicité et sa flexibilité.

I.3.2 Description de l'algorithme

La résolution de fonction telle que sinus ou cosinus par l'algorithme du CORDIC s'appuie sur une méthode de rotation de vecteur dans le plan cartésien.

Dans la technique CORDIC, la rotation plane d'un angle α est atteinte en décomposant l'angle ciblé en plusieurs angles élémentaires.

$\alpha = \sum_{i=0}^{M-1} \delta_i.\theta_i$, $\theta_i = tg^{-1}(2^{-i})$ ....(12)

avec:

M étant la longueur du mot et $\delta i = \pm 1$.

L'indice δi indique le sens de rotation de l'angle pour chaque itération. Cet indice est déterminé à chaque itération selon le résultat d'une comparaison. $\delta_i = Sign(E_i)$ ......(13)

Dans un plan élémentaire en deux dimensions supposons la rotation du vecteur V(x, y) d'un angle α tel qu'illustré à la figure (4).

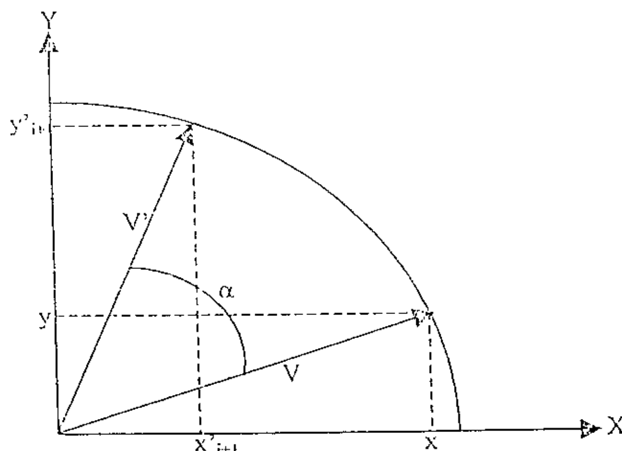


Figure 4. Rotation d'un vecteur V d'un angle α

les coordonnées du vecteur V' sont exprimées selon les équations.

$$x_{i+1} = x_i.\cos(\theta_i) + \delta_i.y_i.\sin(\theta_i)$$
$$y_{i+1} = -\delta_i.x_i.\sin(\theta_i) + y_i.\cos(\theta_i) \quad ....(14)$$

En appliquant la condition $tg(\theta_i) = 2^{-i}$ qui est uniquement valable pour les angles élémentaires, l'équation (13) s'écrira comme suit :

$$x_{i+1} = \cos(\theta_i).(x_i + \delta_i.y_i.2^{-i}).$$
$$y_{i+1} = \cos(\theta_i).(-\delta_i.x_i.2^{-i} + y_i). \quad ....(15)$$

En omettant les termes du cosinus nous obtenons:

$$x'_{i+1} = x'_i + \delta_i . y'_i . 2^{-i}$$
$$y'_{i+1} = -\delta_i . x'_i . 2^{-i} + y'_i \qquad \text{..........(16)}$$

$$E_{i+1} = E_i - \delta_i . Arctg(2^{-i}) \quad \text{avec} \quad E_0 = \alpha \quad \text{et} \quad \delta_i = sign(E_i)$$

Si M est le nombre de pas d'itérations nous aurons:

$x'_M = \zeta_M . x_M$ et $y'_M = \zeta_M . y_M$
où:

$$\zeta_M = \frac{1}{(\prod_{i=0}^{M-1} \cos\theta_i)}.$$

Pour un nombre croissant d'itération, le produit tend vers la valeur de 0.6073. Ainsi le facteur ζ_M qui représente le gain de l'algorithme de rotation est approximativement égale à 1.647.

Soient $(x'_0, y'_0) = (P_{re} - Q_{re}, P_{im} - Q_{im})$ et $\alpha = k.\theta$, en exploitant les équations (15) nous obtenons les résultats suivants:

$$x'_M = \zeta_M (P_{re} - Q_{re}).\cos(k\theta) + \zeta_M (P_{im} - Q_{im}).\sin(k\theta)$$
$$y'_M = -\zeta_M (P_{re} - Q_{re}).\sin(k\theta) + \zeta_M (P_{im} - Q_{im}).\cos(k\theta) \qquad \text{...........(17)}$$

Les expressions des équations (17) sont identiques aux deux dernières équations de la série d'équations (11) qui décrivent l'opération du Papillon a l'exception du facteur ζ_M.

D'après la figure (3), l'opération du papillon ayant un terme de gain constant s'écrit:

$$R'_{re} = \zeta_M (P_{re} + Q_{re}).$$
$$R'_{im} = \zeta_M (P_{im} + Q_{im}).$$

$$\text{.............(18)}$$

$$S'_{re} = \zeta_M (P_{re} - Q_{re}).\cos(k\theta) + \zeta_M (P_{im} - Q_{im}).\sin(k\theta)$$
$$S'_{im} = -\zeta_M (P_{re} - Q_{re}).\sin(k\theta) + \zeta_M (P_{im} - Q_{im}).\cos(k\theta).$$

Le CORDIC possède deux modes de fonctionnement (rotation et vecteur) que nous aborderons en détails lors du chapitre -III-.

I.4 Conclusion

Nous venons de voir dans ce chapitre une technique rapide et efficace nous permettant le calcul de la TFD (*Transformée de Fourier Discrète*) tout en réduisant la complexité temporelle. Cette technique est basée sur des algorithmes connus sous le nom de Transformée de Fourier Rapide TFR ou FFT(*Fast Fourier Transform*).

Le calcul de la FFT n'est autre qu'une série d'opérations élémentaires appelées le Papillon (Butterfly). La réalisation de cette opération fait appel a l'algorithme CORDIC.

Dans le prochain chapitre nous introduirons un des outils de CAO électronique qui est le VHDL .

Chapitre 2

Le VHDL et la technologie FPGA.

II.1 Le VHDL 15

II.1.1 Introduction 15

II.1.2 Principes généraux 16

II.1.2.a Description descendante : le « top down design » ---------- 16
II.1.2.b Les niveaux d'abstraction ---------- 17
II.1.2.c Simulation et/ou synthèse ---------- 18
II.1.2.d L'extérieur de la boîte noire : une « ENTITE » ---------- 20
II.1.2.e Le fonctionnement interne : une « ARCHITECTURE » ---------- 21

II.2 La technologie FPGA 25

II.2.1 Introduction 25

II.2.2 Les LCA de XILINX 25

II.2.3 Les cellules logiques de base des LCA 29

II.2.4 Les différents types d'interconnexions 31

II.3 Convention de XILINX sur les FPGA 31

II.4 Conclusion 32

II - . Le VHDL & la technologie FPGA:

II.1 Le VHDL

II.1.1 Introduction [9, 10, 11]

Le langage VHDL est un langage de description de matériel électronique .Il a été défini et introduit dans les outils de CAO électronique pour apporter de la méthode et de la rigueur dans le cycle de développement des systèmes matériels.

Ce langage est issu du programme VHSIC(Very High Speed Integrated Circuit) initié par le DOD aux Etats-Unis en 1980. Le besoin d'un langage moderne et standard s'est fait ressentir devant l'augmentation importante de la complexité des systèmes électroniques et surtout des coûts de maintenance en résultant .Ce langage est devenu standard international IEEE en 1987.

VHDL est l'abréviation de « Very high speed integrated circuits Hardware Description Langage ». L'ambition des concepteurs du langage est de fournir un outil de description homogène des circuits, qui permette de créer des modèles de simulation et de « compiler » le silicium à partir d'un programme unique. Initialement réservé au monde des circuits numériques, VHDL est en passe d'être étendu aux circuits analogiques.

Deux des intérêts majeurs du langage sont :

❖ *Des niveaux de description très divers*: VHDL permet de représenter le fonctionnement d'une application tant du point de vue système que du point de vue circuit, en descendant jusqu'aux opérateurs les plus élémentaires. A chaque niveau, la description peut être structurelle (portrait des interconnexions entre des sous-fonctions) ou comportementale (langage évolué).

❖ *Son aspect « non propriétaire »*: le développement des circuits logiques a conduit chaque fabriquant à développer son propre langage de description. Le VHDL est en passe de devenir le langage commun à de nombreux systèmes de CAO, indépendants ou liés à des producteurs de circuits, des outils relativement simples d'aide à la programmation des PALs aux ASICs, en passant par les FPGAs.

La description qui suit est loin d'être exhaustive, VHDL est un langage important. Nous en présentons un sous-ensemble qui, nous l'espérons, doit permettre à un néophyte d'aborder ses premières réalisations avec un bagage minimum, limité à des *constructions synthétisables* et, en principe, portable sur n'importe quel compilateur.

II.1.2 Principes généraux [7, 8, 9, 10, 11, 12]

II.1.2.a Description descendante : le « top down design »

Une application un tant soit peu complexe est découpée en sous-ensembles qui échangent des informations suivant un protocole bien défini. Chaque sous-ensemble est, à son tour, divisé et ainsi de suite jusqu'aux opérateurs élémentaires.

Un système est construit comme une hiérarchie d'objets, les détails de réalisation se précisant au fur et à mesure que l'on descend dans cette hiérarchie. A un niveau donné de la hiérarchie, les détails de fonctionnement interne des niveaux inférieurs sont *invisibles*, C'est le principe même de la programmation structurée.

Plusieurs réalisations d'une même fonction pourront être envisagées, sans qu'il soit nécessaire de remettre en cause la conception des niveaux supérieurs ; plusieurs personnes pourront collaborer à un même projet, sans que chacun ait à connaître tous les détails de l'ensemble.

La conception descendante consiste à définir le système en partant du sommet de la hiérarchie, en allant du général au particulier. VHDL permet, par exemple, de tester la validité de la conception d'ensemble, avant que les détails des sous fonctions ne soient complètement définis. A titre d'exemple, l'architecture générale d'un processeur peut être évaluée sans que le mode de réalisation de ses registres internes ne soit connu, le fonctionnement des registres en question sera alors décrit au niveau comportemental.

II.1.2.b Les niveaux d'abstraction

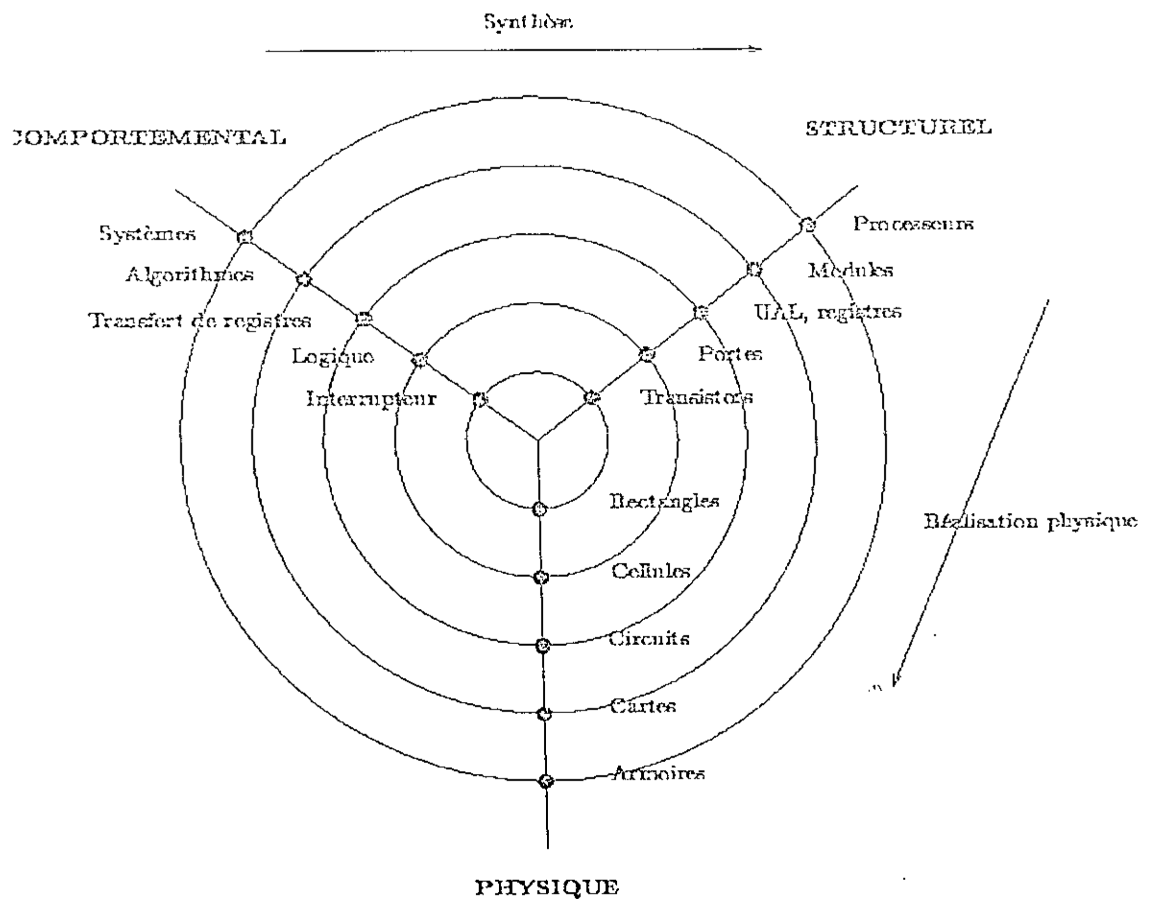


Figure 5. Diagramme en Y introduit par Gajski décrivant les trois vues d'une architecture

Différents niveaux d'abstractions peuvent être répertoriés: interrupteur, logique, transfert de registres, algorithmique et système (fig.5).

Niveau interrupteur(circuit):

Une description au niveau interrupteur manipule les composants de très bas niveau caractérisés par des délais.

Niveau logique:

Une spécification logique est un ensemble de fonctions booléennes ou de machines d'états.

Niveau transfert de registre:

Le niveau supérieur est appelé transfert de registres ou RTL(Register Transfer Level). La description porte sur les états des registres et les transferts d'états. On distingue trois grandes parties: la partie chemin de données qui représente les calculs, la partie contrôle et la partie mémoire.

Niveau algorithmique:

Le niveau algorithmique est le niveau le plus abstrait pour décrire un circuit.

Niveau système:

Le niveau de description le plus haut est le niveau système. Il spécifie un système de façon formelle. On doit souvent passer par plusieurs étapes pour obtenir une spécification claire et correcte.

Travailler à un niveau très abstrait permet justement de vérifier des fonctionnalités et de corriger des erreurs plus facilement.

Au niveau le plus abstrait, la description consiste en un ensemble d'entités fonctionnelles et les relations qui existent entre elles.

II.1.2.cSimulation et/ou synthèse

VHDL a été, initialement, conçu comme un langage de simulation, il est fortement marqué par cet héritage informatique, ce qui est parfois déroutant pour l'électronicien, proche du matériel et qui n'est pas toujours un spécialiste des langages de programmation. Citons quelques exemples :

- Contrairement à C ou PASCAL, VHDL est un langage qui comprend le « parallélisme », c'est à dire que des blocs d'instructions peuvent être exécutés simultanément, et non séquentiellement comme dans un langage procédural traditionnel. Autant ce parallélisme est fondamental pour comprendre le fonctionnement d'un simulateur logique et peut être déroutant pour un programmeur habitué au déroulement séquentiel des instructions qu'il écrit, autant il est évident que le fonctionnement d'un circuit ne dépend pas de l'ordre dans lequel ont été établies les connexions. L'utilisateur de VHDL gagnera beaucoup en ne se laissant pas enfermer dans l'aspect langage de programmation, en se souvenant qu'il est en train de créer un vrai circuit. Les parties séquentielles du langage, car il y en a, doivent, dans ce contexte, être comprises soit comme une facilité offerte dans l'écriture de certaines fonctions, soit comme le moyen de décrire des opérateurs fondamentalement séquentiels : les opérateurs synchrones.

– La modélisation correcte d'un système suppose de prendre en compte, au niveau du simulateur, les imperfections du monde réel. VHDL offre donc la possibilité de spécifier des retards, de préciser ce qui se passe lors d'un conflit de bus etc. Pour simuler toutes ces vicissitudes, le langage offre toute une gamme d'outils : signaux qui prennent une valeur inconnue, messages d'erreurs quand un « circuit » détecte une violation de *set-up time*, changements d'états retardés pour simuler les temps de propagation. Toutes les constructions associées de ce type ne sont évidemment *pas* synthétisables. La difficulté principale est que, suivant les compilateurs, la frontière entre ce qui est synthétisable et ce qui ne l'est pas n'est pas toujours la même, même pour des compilateurs qui respectent la norme IEEE-1076. Avant d'utiliser un outil de synthèse, le concepteur de circuit a tout à gagner à lire très attentivement la présentation du sous-ensemble de VHDL accepté par cet outil.

– Trois classes de données existent en VHDL : les constantes, les variables et les signaux. La nature des signaux ne présente aucune ambiguïté, ce sont des objets qui véhiculent une information logique tant du point de vue simulation que dans la réalité. Les signaux qui ont échappé aux simplifications logiques, apportées par l'optimiseur toujours présent, sont des vraies équipotentielles du schéma final. Les variables sont destinées, comme dans tout langage, à stocker temporairement des valeurs, dans l'optique d'une utilisation future, sans chercher à représenter la réalité.

Certains compilateurs considèrent que les variables n'ont aucune existence réelle, au niveau du circuit, qu'elles ne sont que des outils de description fonctionnelle. D'autres transforment, éventuellement (cela dépend de l'optimiseur), les variables en cellules mémoires...

II.1.2.d L'extérieur de la boîte noire : une « ENTITE »

Nous avons mentionné que, dans une construction hiérarchique, les niveaux supérieurs n'ont pas à connaître les détails des niveaux inférieurs. Une fonction logique sera vue, dans cette optique, comme un assemblage de « boîtes noires », dont, syntaxiquement parlant, seules les modes d'accès sont nécessaires à l'utilisateur.

La construction qui décrit l'extérieur d'une fonction est l'entité (*entity*). La déclaration correspondante lui donne un nom et précise la liste des signaux d'entrée et de sortie :

```
entity mon_circuit is port (
entree1 : in bit;
entree2, entree3 : in bit;
entree4 : in bit_vector(0 to 3);
sortie1, sortie2 : out bit;
sortie3 : out bit_vector(0 to 5))
end mon_circuit;
```

Dans l'exemple qui précède, les noms des objets, qui dépendent du choix de l'utilisateur, sont écrits en italique, les autres mots sont des mots-clés du langage. Les choix possibles pour le sens de transfert sont : in, out, inout et buffer (une sortie qui peut être « lue » par l'intérieur du circuit).

Les choix possibles pour les types de données échangées sont les mêmes que pour les signaux (voir ci-dessous).

II.1.2.c Le fonctionnement interne : une « ARCHITECTURE »

L'architecture décrit le fonctionnement interne d'un circuit auquel est attaché une entité. Ce fonctionnement peut être décrit de différentes façons :

Description structurelle : le circuit est vu comme un assemblage de composants de niveau inférieur, c'est une description « schématique ». Souvent ce mode de description est utilisé au niveau le plus élevé de la hiérarchie, chaque composant étant lui-même défini par un programme VHDL (entité et architecture).

Description comportementale : le comportement matériel du circuit est décrit par un algorithme, indépendamment de la façon dont il est réalisé au niveau structurel.

Description par un flot de données : le fonctionnement du circuit est décrit par un flot de données qui vont des entrées vers les sorties, en subissant, étape par étape, des transformations élémentaires successives. Ce mode de description permet de reproduire l'architecture logique, en couches successives, des opérateurs combinatoires.

Le flot de données et la représentation comportementale sont très voisins, dans les deux cas le concepteur peut faire appel à des instructions de haut niveau. La première méthode utilise un grand nombre de signaux internes qui conduisent au résultat par des transformations de proche en proche, la seconde utilise des blocs de programme (les processus explicites), qui manipulent de nombreux signaux avec des algorithmes séquentiels.

La syntaxe générale d'une architecture comporte une partie de déclaration et un corps de programme :

```
architecture exemple of mon_circuit is
partie déclarative optionnelle : types, constantes,
signaux locaux, composants.
begin
corps de l'architecture.
suite d'instructions parallèles :
affectations de signaux;
processus explicites;
blocs;
```

Des algorithmes séquentiels décrivent un câblage parallèle : les « PROCESSUS »

« Un processus est une instruction *concurrente* (deux instructions concurrentes sont simultanées) qui définit un comportement qui doit avoir lieu quand ce processus devient actif. Le comportement est spécifié par une suite d'instructions *séquentielles* exécutées dans le processus. »

Que cela signifie-t-il ?

Trois choses :

1)- Les différentes parties d'une réalisation interagissent simultanément, peu importe l'ordre dans lequel un câbleur soude ses composants, le résultat sera le même. Le langage doit donc comporter une contrainte de « parallélisme » entre ses instructions. Cela implique des différences notables avec un langage procédural comme le C.

En VHDL :

```
a <= b ;
c <= a + d ;
```

et

```
c <= a + d ;
a <= b ;
```

représentent la même chose, ce qui est notablement différent de ce qui se passerait en C pour :

```
a = b ;
c = a + d ;
```

```
et
c <= a + d ;
a <= b ;
```

Les affectations de signaux, à *l'extérieur* d'un processus explicite, sont traitées comme des processus tellement élémentaires qu'il est inutile de les déclarer comme tels. Ces affectations sont traitées en parallèle, de la même façon que plusieurs processus indépendants.

2)- L'algorithmique fait grand usage d'instructions séquentielles pour décrire le monde. VHDL offre cette facilité à *l'intérieur* d'un processus explicitement déclaré. Dans le corps d'un processus il sera possible d'utiliser des variables, des boucles, des conditions, dont le sens est le même que dans les langages séquentiels.

Même les affectations entre signaux sont des instructions séquentielles quand elles apparaissent à l'intérieur d'un processus. Seul sera visible de l'extérieur le résultat final obtenu à la fin du processus.

3)- Les opérateurs séquentiels, surtout synchrones, mais pas exclusivement eux, comportent « naturellement » la notion de mémoire, qui est le fondement de l'algorithmique traditionnelle. Les processus sont *la* représentation privilégiée de ces opérateurs. Mais attention, *la réciproque n'est pas vraie*, il est parfaitement possible de décrire un opérateur purement combinatoire par un processus, le programmeur utilise alors de cet objet la seule facilité d'écriture de l'algorithme.

Outre les simples affectations de signaux, qui sont en elles mêmes des processus implicites à part entière, la description d'un processus obéit à la syntaxe suivante :

Processus : syntaxe générale

```
[étiquette : ] process [ (liste de sensibilité) ]
partie déclarative optionnelle : variables notamment
begin
corps du processus.
```

```
instructions séquentielles
end process [ étiquette ] ;
```

Les éléments mis entre crochets sont optionnels, ils peuvent être omis sans qu'il y ait d'erreur de syntaxe.

La liste de sensibilité est la liste des signaux qui déclenchent, par le changement de valeur de l'un d'entre eux, l'activité du processus. Cette liste peut être remplacée par une instruction « wait » dans le corps du processus :

L'instruction `wait`

Cette instruction indique au processus que son déroulement doit être suspendu dans l'attente d'un événement sur un signal (le changement de valeur du signal), tant qu'une condition n'est pas réalisée.

Sa syntaxe générale est :

```
wait [on liste_de_signaux ] [until condition ] ;
```

La liste des signaux dont l'instruction attend le changement de valeur joue exactement le même rôle que la liste de sensibilité du processus, mais *l'instruction* `wait` *ne peut pas être utilisée en même temps qu'une liste de sensibilité*. La tendance, pour les évolutions futures du langage, semble être à la suppression des listes de sensibilités, pour n'utiliser que les instructions d'attente.

II.2 La technologie FPGA

II.2.1 Introduction

Nous allons vous présenter dans ce chapitre les circuits FPGA par leurs fabricants ,et nous allons nous intéresser surtout aux FPGA de type RAM, appelés LCA par XILINX.

II.2.2 Les LCA de XILINX [5, 6]

Les réseaux logiques programmables de types RAM ou FPGA de type RAM, que nous appelons LCA , ont été inventés par la firme américaine XILINX et ont été commercialisés pour la première fois en 1984.

Pour concevoir ses produits, XILINX a étudié les divers types de circuits programmables disponibles sur le marché a l'époque afin de déterminer leurs limitations. L'approche classique des PLD, c'est a dire des PAL, GAL et EPLD conduit, selon XILINX, a un gaspillage relativement important de ressources.

En effet même si un boîtier comporte N portes, un grand nombre d'entre elles sont perdues du fait de leur non utilisation par la matrice d'interconnexion comme le montre la figure (6).

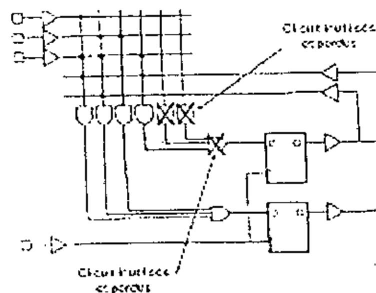


Figure 6. mise en évidence du gaspillage de ressources dans un PLD classique.

Ceci tient essentiellement au fait que, si la structure d'entrée des portes perdues est bien programmable et offre donc toute la souplesse voulue, la structure de sortie est fixe. Les portes sont connectées à la macro-cellule uniquement. Si cette dernière n'en a pas besoin, les portes concernées restent donc inutilisées.

La solution idéale, en termes d'utilisation des ressources, est celle proposée par les réseaux de portes programmables par masque .

En effet, dans cette approche, la puce de silicium est entourée de blocs d'entées/sorties et son centre est constitué par une matrice de portes élémentaires. Lors de la phase de masquage, ces portes sont interconnectées à la demande étant donné que sur la puce vierge il n'existe aucune connexion préétablie. Le taux d'utilisation des ressources est ainsi excellent mais ce type de circuit n'est programmable que par le fabriquant avec tous les inconvénients que l'on sait (coûts, délais, minimum de pièces identiques à commander, outils de développements très lourds à manier et très coûteux).

XILINX a donc en quelque sorte combiné ces deux approches pour proposer son architecture LCA. Il est à noter dés à présent qu'ALTERA a procédé à la même analyse quant au gaspillage de ressources des structures de PLD classiques ce qui l'a conduit à proposer ses circuits FLEX.

Comme le montre la figure (7), l'organisation générale ressemble à celle des réseaux des portes programmables par masque puisque l'on retrouve sur la périphérie de la puce des blocs d'entrées/sorties programmables. Par contre, les portes élémentaires centrales ont été remplacées par ce que XILINX appelle des CLB pour (Configurable Logic Bloc) que l'ont peut assimiler en première approximation aux macro-cellules des circuits logiques programmables type EPLD.

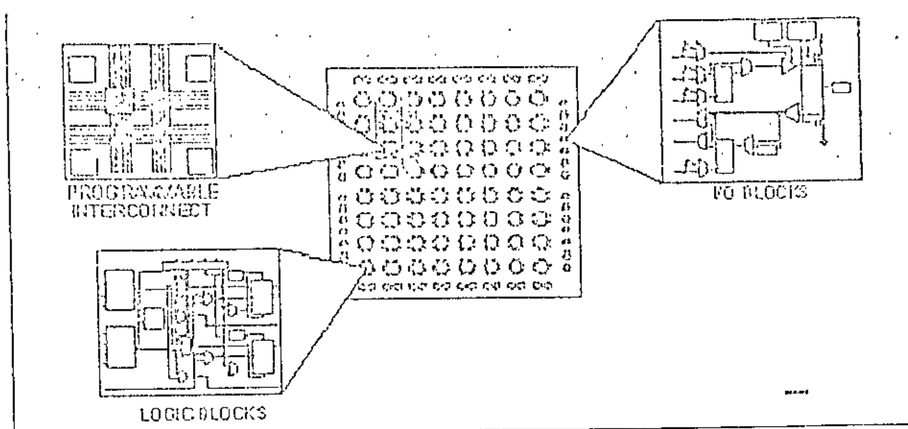


Figure 7. la structure adoptée par XILINX pour ces LCA.

Cette utilisation de CLB permet de réduire le nombre d'interconnexions nécessaires entre les différents blocs par rapports aux portes élémentaires que l'on trouve dans les réseaux programmables par masque. Mais le point fort des LCA ne se limite pas à cela ; en effet, nous n'avons pas encore expliqué comment était résolu ce fameux problème d'interconnexion et c'est cela qui fait toute l'originalité et toute la souplesse d'utilisation des LCA.

En fait , il existe dans tout LCA des matrices de lignes d'interconnexions qui sillonnent horizontalement et verticalement les divers CLB. Les connexions sur ces lignes sont effectuées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM.

Il faut savoir que sous le réseau logique proprement dit se trouve une zone de mémoire dans laquelle est chargée la configuration de connexions a établir.

Une RAM étant une mémoire dans laquelle on peut lire ou écrire autant de fois que nécessaire et sans restriction ; on conçoit que les LCA soient ainsi programmables et effaçables électriquement, mais comme ces opérations sont en fait de simples écritures mémoire, elle ne prennent que quelques millisecondes et ne font appel a aucun programmateur spécialisé. Elles n'utilisent, en effet, aucune autre tension ou tension spéciale et se contentent de l'alimentation 5 volts du circuit lui même.

Les LCA étaient donc les premiers composants logiques programmables en circuit et si l'on se remémore ce qui existait en 1984 ou quasiment personne n'envisageait cette possibilité, cela représentait une véritable petite révolution. Seul petit bémol, cette possibilité n'était pas due à une technologie CMOS, pas encore au point pour ce faire à l'époque, mais à la présence de RAM statique dans le LCA ce qui induisait quelques petites contraintes.

En effet, comme vous le savez certainement, une RAM perd son contenu lors de toute coupure d'alimentation. Notre LCA, si l'on ne prend pas de précautions particulières, va donc perdre sa configuration de connexions lors de chaque arrêt du système. Pour remédier à cela trois solutions sont envisageables :

L'utilisation d'un support spécial intégrant une petite pile au lithium qui maintient le LCA alimenté même en cas de coupure secteur. Comme la consommation du plan mémoire est dérisoire au repos, la durée de vie de la pile peut être de l'ordre de 3ans.

L'utilisation d'une mémoire morte, de type PROM qui, lors de chaque mise sous tension, se recopie en quelque millisecondes dans le LCA et le reconfigure ainsi immédiatement. Cette façon de faire est évidemment nettement préférable à la précédente, car il n'y a plus à craindre de perte d'information suite à l'épuisement de la pile.

L'utilisation des ressources d'un microprocesseur ou microcontrôleur se trouvant dans le même appareil que le LCA et qui peut alors télécharger une partie de sa mémoire dans le LCA pour le configurer lors de chaque mise sous tension.

On le voit, du fait de cette utilisation d'une RAM pour configurer les connexions, le LCA est un peu contraignant pour ce qui est de la phase de mise en marche. Par contre, il ouvre des horizons nouveaux aux concepteurs de circuits car, du fait de cette RAM qui peut être lue mais surtout écrite a tout instant, il est possible de modifier la configuration de la logique réalisée par le circuit alors même que le montage qui l'utilise est en fonctionnement.

II.2.3. Les cellules logiques de base des LCA [5, 6]

Comme nous l'avons laissé entendre précédemment, les LCA utilisent deux types de cellules de base :

Les cellules d'entrées/sorties, appelées ici IOB pour Input Output Block.
Les cellules logiques proprement dites appelées CLB pour Configurable Logic Block.

Voyons tout d'abord ce qu'il y a à l'intérieur d'un IOB grâce à la figure (8) qui présente de façon aussi schématique que possible cette structure qui peut être entrée, sortie ou entrée/sortie.

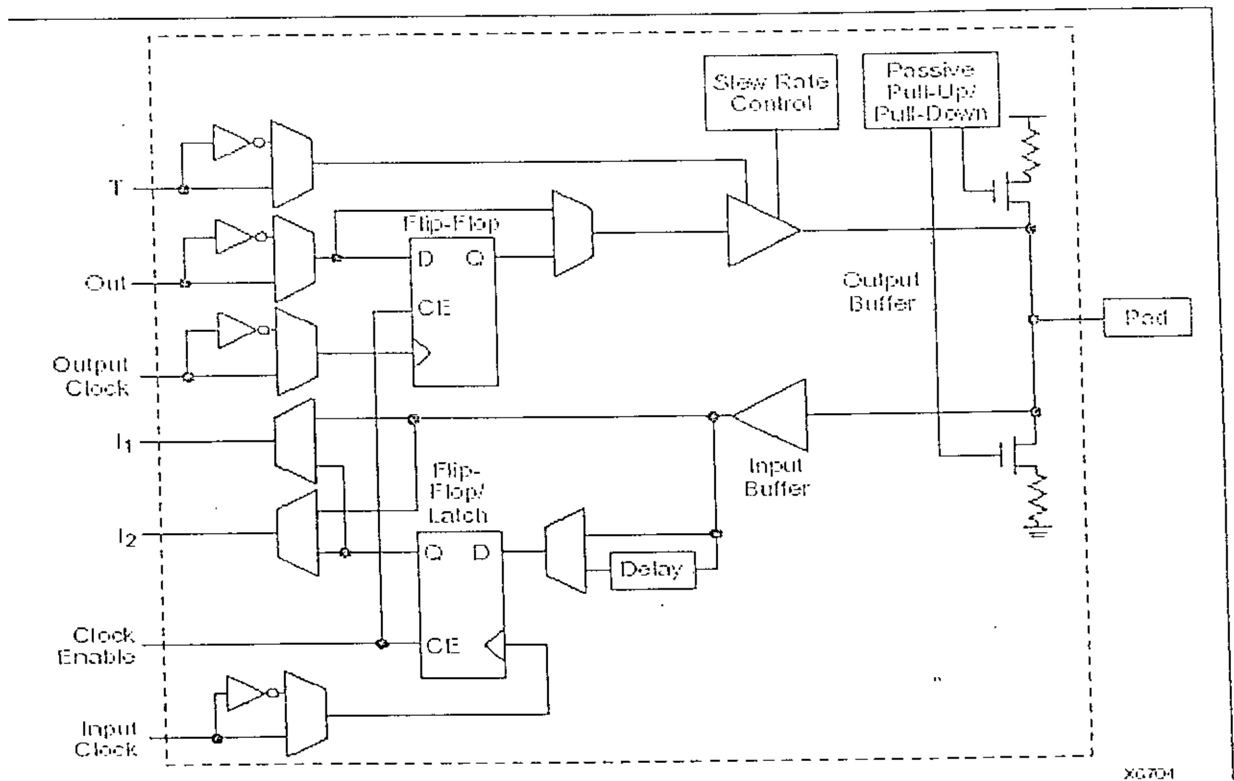


Figure 8. structure detaillée d'un IOB.

Nous voyons tout d'abord en partie haute cinq rectangles qui ne correspondent pas à des connexions au sens propre du terme, mais à des sélections de modes de fonctionnement. Ces sélections sont cependant programmables et configurables avec la même souplesse que les connexions et font bien évidemment appel aux cellules de RAM que nous venons d'évoquer. On dispose des possibilités suivantes :

- inversion ou non du signal avant son application à l'IOB.
- Inversion ou non du signal de commande du buffer de sortie qui est du type trois états.
- Sélection du type de sortie avec ou sans intervention du registre situé en partie haute de la figure (9).
- Sélection du slew rate ou temps de montée du signal de sortie pour s'adapter à la logique connectée au LCA.
- Mise en place d'un pull-up passif ou résistance de tirage au niveau haut lorsque la patte est en entrée(ce qui permet de la laisser en l'air tout en fixant son niveau comme dans tout bon circuit intégré qui se respecte).

Ceci étant vu, le signal de sortie peut passer ou non par une bascule de type D. cette dernière est remise à zéro par un signal de reset global qui est activé automatiquement lors de la phase de configuration du circuit permettant ainsi un démarrage dans un état parfaitement défini. Ce reset peut également être activé par une patte dédiée du boîtier.

Le signal d'entrée, quant à lui, peut passer ou non au travers d'une bascule du même type qui dispose des mêmes possibilités de connexions et de travail que la bascule de sortie.

Comme nous pouvons le constater, ce bloc d'entrées/sorties ou IOB dispose donc de toutes les fonctionnalités souhaitables quelle que soit l'application à laquelle on le destine.

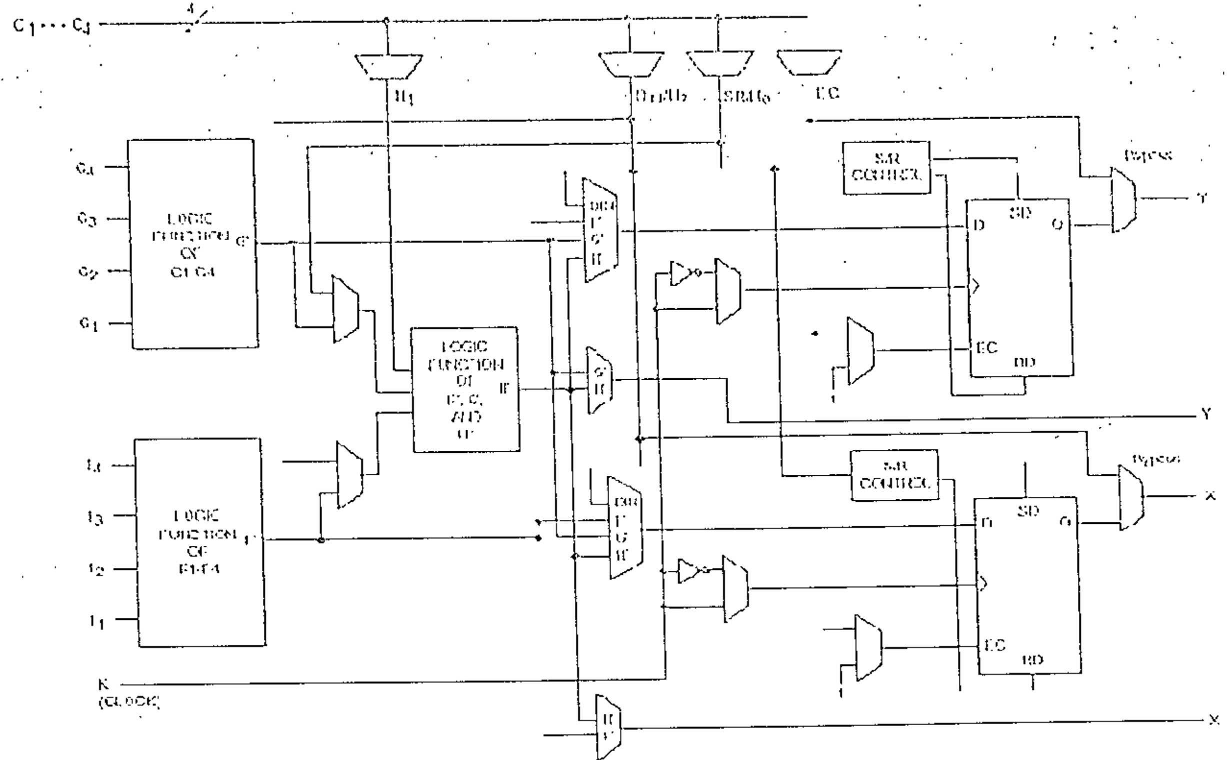


Figure 9. Structure detaillée d'un CLB.

La cellule logique ou CLB n'est pas en reste comme vous pouvez déjà le vérifier à l'examen de sa structure présentée figure(9). Elle est composée d'un certain nombre de sous-ensembles de base qui sont :

Deux bascules de type D.
Un bloc de logique combinatoire.
Divers multiplexeurs utilisés comme commutateurs pour sélectionner les divers modes de fonctionnement de la cellule.

II.2.4 Les différents types d'interconnexions [5, 6]

Il ne suffit pas de disposer de blocs logiques ou d'entrées/ sorties performants, encore faut-il pouvoir les interconnecter avec un maximum d'efficacité afin que leur taux d'utilisation dans un circuit donné soit aussi élevé que possible.

Pour y parvenir, XILINX propose trois types de moyens d'interconnexions différents que nous allons décrire maintenant. Bien sur, tous reposent sur l'établissement ou non de liaisons avec des transistors MOS pilotés par de la mémoire RAM, puisque c'est le concept de base des LCA mais, selon la longueur et la destination des liaisons, les cheminements utilisés ne sont pas les mêmes ; on distingue en effet :

- Les interconnexions à usage général.
- Les interconnexions directes.
- Les lignes longues (long lines).

Tous ces chemins d'interconnexions sont accessibles au niveau des transistors MOS de commutation matérialisés par ce que XILINX appelle les PIP(Programmable Interconnect Points) répartis sur toute la puce comme le montre un extrait du dessin de celle-ci visible figure(10).

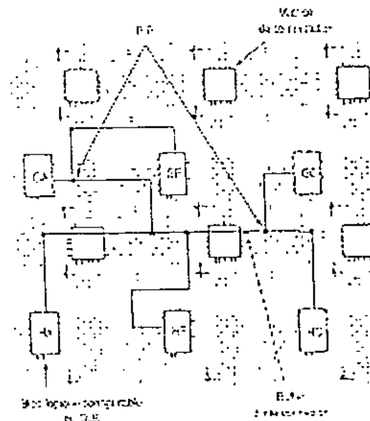


Figure 10. Mise en évidence de quelques PIP

Les interconnexions à usage général consistent en une grille de cinq segments métalliques verticaux et de cinq segments métalliques horizontaux placés entre les rangées et les colonnes de CLB et d'IOB.

Chaque segment fait la hauteur ou la largeur d'un bloc logique. A chaque intersection de rangée et de colonne se trouve placée une matrice de commutation qui permet de raccorder les segments entre eux selon diverses configurations présentées figure (11).

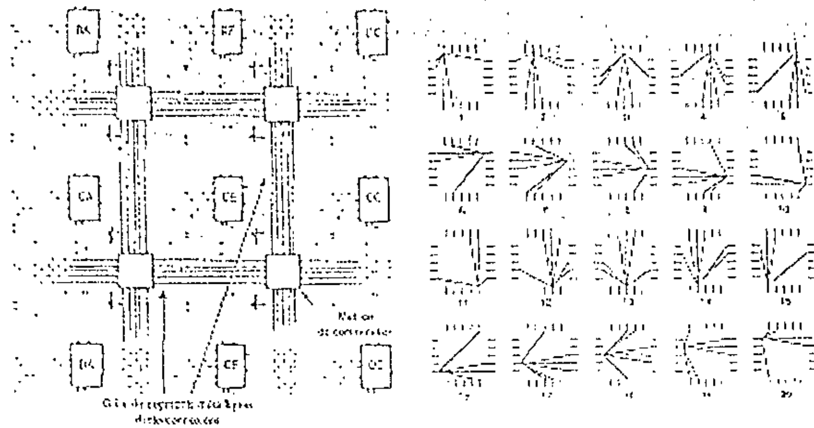


Figure 11. Emplacements et possibilités des matrices de commutation

Même si toutes les combinaisons ne sont pas autorisées, le nombre de variantes disponibles permet déjà une grande liberté.

Afin que les signaux qui parcourent de grandes distances via ces lignes sur la puce ne soient pas trop affaiblis, des buffers sont implantés régulièrement en haut et à droite de toutes les matrices de commutation. Il est alors possible d'y faire passer les signaux qui le nécessitent.

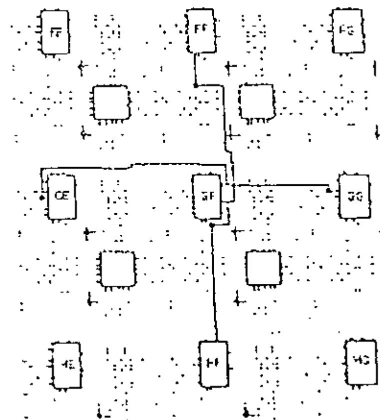


Figure 12. Utilisation des possibilités d'interconnexions directes.

Les interconnexions directes, dont le principe est visible figure (12) permettant l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en termes de vitesse et d'occupation de la puce. Elles résultent du fait que, en raison de la géométrie des cellules, il est possible de relier directement certaines entrées de l'une aux sorties de l'autre sans faire appel aux autres ressources d'interconnexions, les liaisons qui peuvent être établies de cette façon sont les suivantes :

Pour chaque CLB, la sortie X peut être reliée directement à l'entrée B du CLB situé immédiatement sur sa droite et à l'entrée C du CLB situé sur sa gauche. La sortie Y, quant à elle, peut être reliée directement de la même façon à l'entrée D du CLB du dessus ou à l'entrée A du CLB de dessous.

Pour chaque CLB adjacent à un IOB, les connexions directes sont possibles tour à tour avec les entrées I ou O des IOB selon leur positionnement sur la puce.

Les lignes longues "long lines" sont des moyens d'interconnexions un peu différents de ce que nous venons de voir en ce sens qu'elles ne passent pas par les matrices de commutations. Ce sont des lignes métalliques qui traversent la puce de part en part, de haut en bas et de droite à gauche. Elles sont utilisées en priorité lorsque les signaux doivent être transmis avec un minimum de retard entre différents éléments afin d'assurer un synchronisme d'action aussi parfait que possible.

La figure (13) montre le positionnement de ces lignes longues qui sont au nombre de trois verticalement entre chaque colonne CLB et de deux horizontalement entre chaque ligne de CLB. Deux lignes longues additionnelles sont placées à coté des matrices de commutation les plus externes et peuvent être scindées en deux parties pour former des demi-lignes longues.

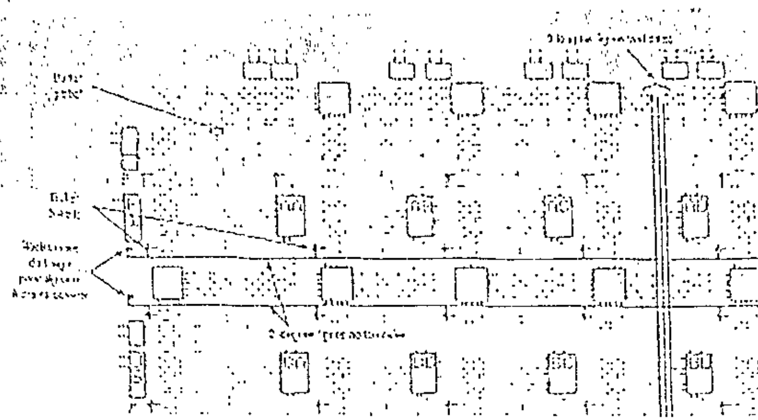
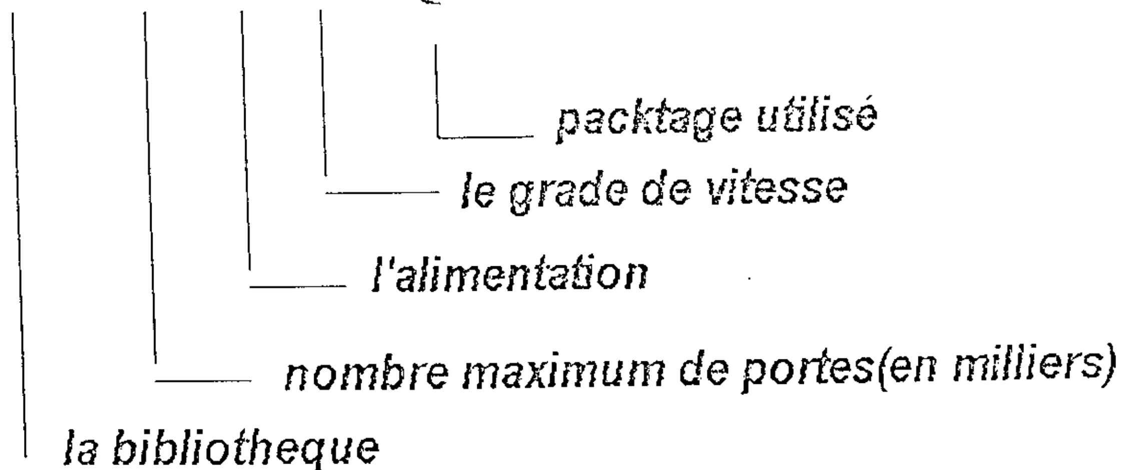


Figure 13. emplacement des longues lignes.

Ces lignes longues peuvent être pilotées par la sortie d'un IOB ou d'un CLB de façon traditionnelle, au moyen d'un buffer disponible, ou bien encore en mode bus par utilisation de buffers trois états également disponibles sur la puce.

II.3 Convention de XILINX SUR LES FPGA



II.4 Conclusion

Ce chapitre a mis en lumière un certain nombre de particularités syntaxiques mais surtout méthodologiques liées à la manière dont le code est écrit.

Finalement, la structure du VHDL est représentative de la manière dont le problème est décomposé et analysé, les clés de son succès réside en sa simplicité, son abstraction et sa modularité.

Il existe plusieurs technologies dans lesquelles sont fabriqués ou programmés les circuits intégrés développés en VHDL. Il est vrai que le but de VHDL et de la synthèse en particulier est de devenir indépendant de la technologie et des contraintes qui y sont liées. Cela reste entièrement vrai, mais il est également envisageable d'orienter un peu la manière dont est conçu et décrit le design pour qu'il s'adapte mieux aux capacités de la cible. Dans notre cas le circuit cible est le FPGA .

Les FPGA sont des circuits intégrés complètement fabriqués et totalement fonctionnels. Nous ne parlons plus ici de créer des masques pour des dépositions ou des dopages, mais exclusivement de programmation de la logique existante.

Notre outil de développement est XILINX FOUNDATION qui englobe l'éditeur VHDL, l'éditeur de schéma, l'éditeur de machine d'états et le simulateur ainsi que d'autres outils.

Chapitre 3

Conception d'architecture et simulation fonctionnelle

III.1 INTRODUCTION..38

III.2 L'UNITÉ DE GÉNÉRATION D'ADRESSES........................39

III.2.1 ARCHITECTURE FONCTIONNELLE....................................41

III.2.2 SIMULATION FONCTIONNELLE ET INTERPRÉTATION DES ÉSULTATS 45

III.3 L'UNITÉ PAPILLON (BUTTERFLY)..................................46

III.3.1 L'UNITÉ SCALE 47

III.3.1.a Architecture fonctionnelle..48

III.3.1.b Simulation fonctionnelle et interprétation des résultats 49

III.3.2 L'UNITE CORDIC 51

III.3.2.a Résolution d'opérations mathématiques par le CORDIC......................51

III.3.2.b Simulations et résultats..51

III.3.2.c Architecture fonctionnelle 56

III.4 L'UNITÉ DE CONTRÔLE 58

III.5 - CONCLUSION 60

III - Conception d'architecture et simulation fonctionnelle.

III.1 Introduction.

La première phase de notre projet consiste à rédiger un document contenant une description textuelle et graphique de nos besoins. Nous analysons alors ce document pour en extraire une architecture. Cette architecture sera éventuellement le fruit d'un compromis entre différentes solutions.

Nous essayerons de détecter très tôt d'éventuelles lacunes ou incohérences dans notre conception pour éliminer au maximum toute zone d'ombre.

Dans la phase suivante, l'architecture générale est découpée en modules. Leur contenu (leur fonction) est décrit soit sous forme de texte VHDL, soit sous forme de schéma de XILINX FOUNDATION ou bien sous forme de machines d'état FSM(Finite State Machine).

Chaque module est alors simulé séparément pour détecter le plus tôt possible un éventuel problème.

Remarquons que le découpage de l'architecture en modules représente une méthode "*top-down*", c'est-à-dire du haut vers le bas. La simulation et la validation quant à elles utilisent une méthode "*bottom-up* "(du bas vers le haut).

Dès que tous les modules sont fonctionnels, nous pouvons les assembler pour constituer le circuit complet. Nous aborderons à ce moment les simulations et validations de haut niveau qui vérifient, normalement, la fonction décrite dans la spécification de besoin.

Une méthodologie rigoureuse, une modularité bien adaptée et des simulations adéquates permettent de bien cerner le fonctionnement des modules numériques .

Le synoptique de notre architecture générale est donné dans la figure suivante :

Figure 14. Synoptique du schéma général

III.2 L'unité de génération d'adresses. [4]

Cette unité génère l'adresse de la donnée sur laquelle sera effectué le calcul réalisé par l'unité Papillon et où le résultat sera écrit ultérieurement, elle calcule également le coefficient (k) nécessaire à l'opération du Papillon (fig.3).

L'unité de génération d'adresses est la conception matérielle de l'algorithme qui a été réalisé à partir de l'organigramme suivant et qui traduit la méthode de calcul représentée par le graphe de la figure 2.

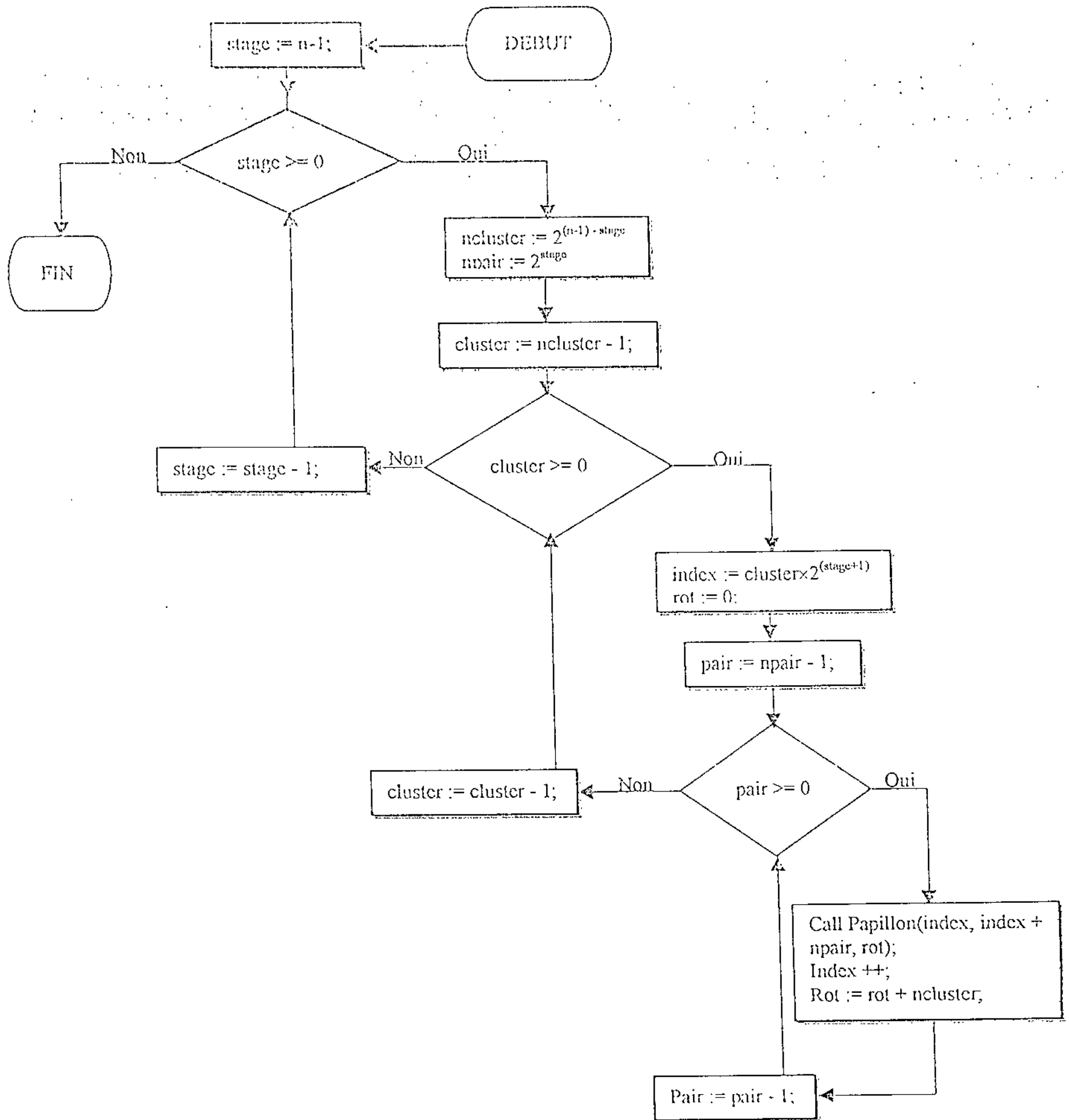


Figure 15. L'organigramme de l'unité de génération d'adresses

A présent, nous allons traduire l'organigramme précédent dans un langage évolué qui est le langage C.

```
/*** Algorithme de génération d'adresses pour le calcul de la FFT pour N = 2^n points
**/
Program Gene_Addr
Var  init: n, stage, nstage, cluster, ncluster, pair, npair, index, rot ;
{
   for (stage = n-1; stage >= 0; stage--)
   {
         ncluster := 2**(n-1-stage);
         npair := 2**stage;
         for (cluster = ncluster -1; cluster >= 0; cluster--)
            {
               index := cluster*2**(stage+1);
              rot := 0;
             for( pair = npair-1; pair >= 0; pair--)
                {
                 Butterfly (index, index + npair, rot);
                 Index ++;
                 Rot := rot + ncluster;
               } /* Fin de la boucle pair */
         } /* Fin de la boucle cluster */
  }  /* Fin de la boucle stage */
} /* Fin du programme */
```

Remarque : Afin de calculer les N points de FFT donnés par : N = 2n = 128 ⇒ n = 7 bits. Nous avons besoin d'un bus d'adresse de 7 bits.

Nous constatons que l'algorithme contient 3 boucles, qui seront représentées sur le schéma par les blocs : *Stage*, *Cluster* et *Pair*.

III.2.1 Architecture fonctionnelle

Dans cette partie nous allons aborder le rôle de chaque entité constituant le module de génération d'adresses, avant d'introduire le schéma général représentant l'algorithme vu lors du paragraphe précédent.

- **Le bloc NPT:** il sert à stoker la valeur (n-1) qui est dans notre cas égale à 6.
- **Le bloc Stage:** le rôle de ce bloc est de réaliser l'instruction [*for (stage = n-1; stage>= 0; stage--*] grâce a un décompteur. Ce dernier est commandé par les signaux d'entrée *CE* (validation du circuit), *Load* (autorisation de chargement) en plus du bus de données (3bits) et des signaux de sortie : *TCs* qui indique la fin du décomptage et la sortie donnée (3bits).
- **Le bloc SUB3 :** ce bloc calcule la valeur [*n-1-stage*] et la transmet au bloc DECO8.

▸ Le bloc DECO8: calcule la valeur [2**(n-1-stage)] et fournit ainsi la valeur *ncluster*; qui est obtenue par un décalage à gauche d'un nombre égal à *n-1-stage*.

▸ Le bloc Cluster: est un décompteur qui réalise la boucle [*for(cluster =ncluster-1; cluster >=0; cluster--)*] le chargement de ce décompteur est commandé par le signal d'entrée *Load_Cluster* et l'autorisation de décomptage est donné par *CE_Cluster*.

▸ Le bloc ICNR3: calcule la valeur [*Stage + 1*].

▸ Le bloc LSHFT: la réalisation de l'instruction [*Index = cluster*2(Stage + 1)*] revient à décaler la valeur *cluster* de *(Stage + 1)*bits à gauche.

▸ Le bloc PAIR: sert à réaliser la boucle [*For(Pair = npair-1; pair>=0; pari--)*]. Notons que *npair* = 2^{*Stage} est réalisée par le bloc DECO8 et *npair-1* par le bloc DECR8.

▸ Le bloc CC8CLE: est destiné au calcul [*Index ++;*].

▸ Le bloc Rot: est un registre contenant la valeur k et son association avec l'additionneur ADD8, nous permet de réaliser l'instruction [*Rot = Rot + ncluster*].

Les résultats présents à la sortie des blocs Angle, Index et Index + NP représentent respectivement le coefficient k, l'adresse de la première et la deuxième donnée nécessaire à l'unité Papillon.

Remarque : Pour de plus amples détails concernant l'architecture interne des différents blocs constituant l'unité de génération d'adresses nous invitons vivement nos lecteurs à consulter l'annexe A.

Le bon fonctionnement de notre unité nous impose une synchronisation entre les blocs (*Stage*, *Cluster*, *Pair* et les registres de sortie) de la manière suivante:

Une impulsion START attaque le bloc *Stage* à travers l'entrée *Load* lui permettant ainsi de charger la valeur présente à la sortie de NPT.
L'impulsion START retardée d'une période d'horloge après un passage par une bascule D attaque le Cluster afin de permettre aux blocs (SUB3, DECO8, DECR8) de générer la valeur *ncluster*.
L'impulsion START est cette fois-ci retardée de deux périodes d'horloge avant d'atteindre le bloc *PAIR* cela permet au bloc *Cluster* de charger la valeur se trouvant à son entrée.
Finalement l'impulsion START est retardée de trois périodes d'horloges cela permet d'achever les calculs effectués par les différents blocs avant de valider les résultats présents aux entrées des registres buffers de sortie.

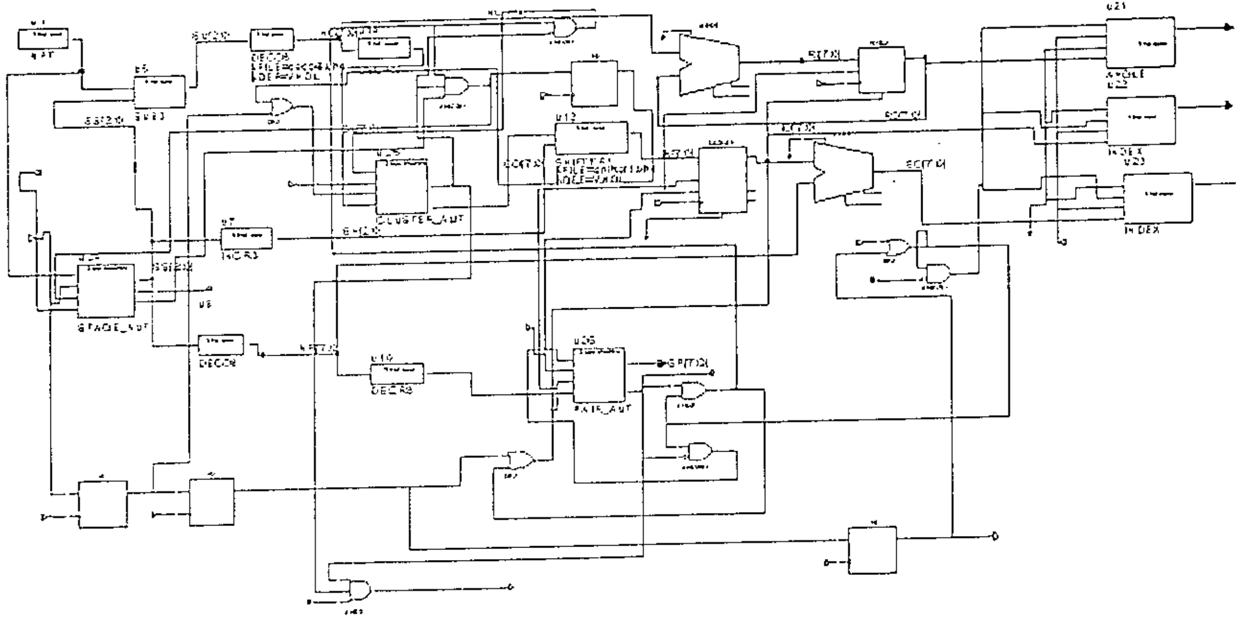


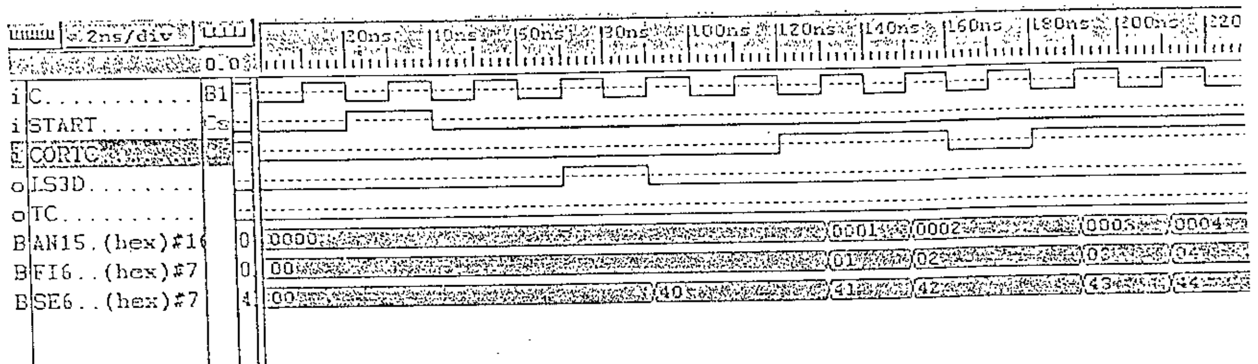
Figure 16. Le bloc de génération d'adresses.

III.2.2 Simulation fonctionnelle et interprétation des résultats

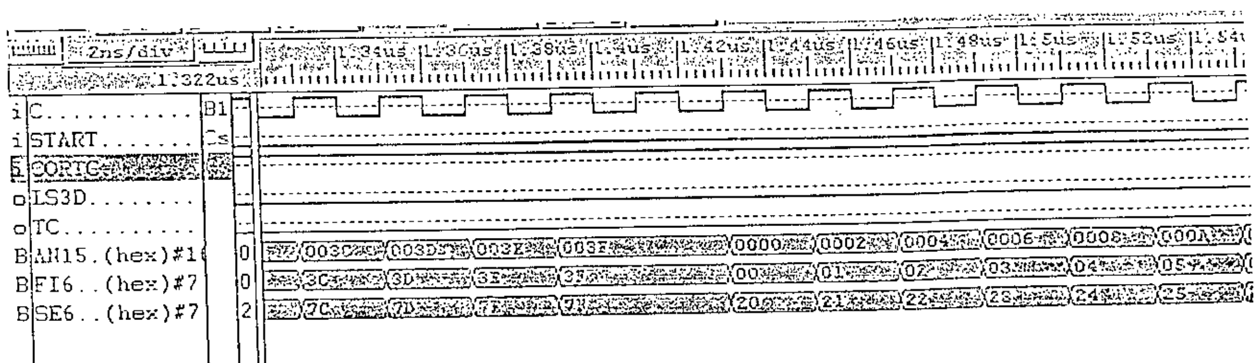
Afin de nous assurer du bon fonctionnement du module de génération d'adresse, nous avons procédé de la manière suivante :

Au début, nous avons commencé par tester tous les blocs qui constituent l'unité de génération d'adresses. Après s'être assuré de leur efficacité, nous les avons interconnectés et par ce fait le schéma de la figure précédente a été obtenu .

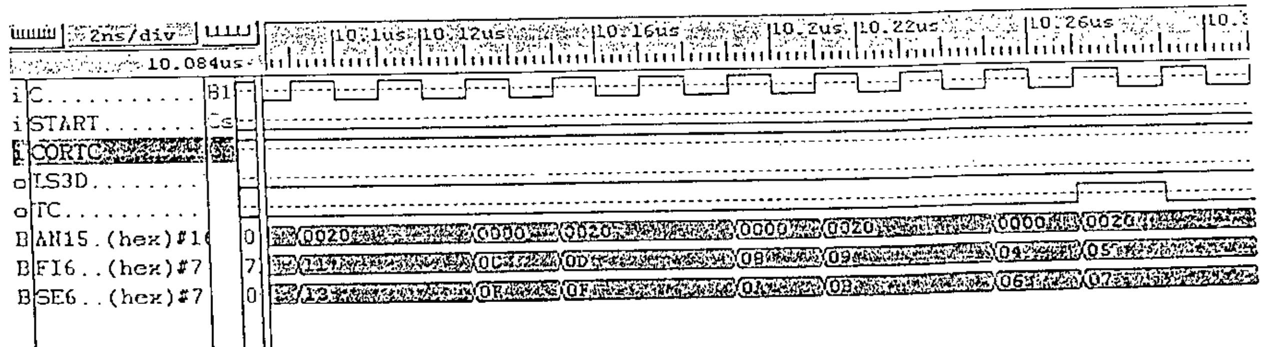
Ensuite nous somme passé à la simulation fonctionnelle du bloc de génération d'adresse qui a la forme suivante :



(a)



(b)



(c)

Figure 17. Simulation de l'unité de génération d'adresse.

Notons que les trois figures (a, b, c) représentent la même simulation mais à des intervalles de temps différents.

De la figure (a) nous remarquons que toutes les sorties sont initialement à zéro, et cela avant l'arrivée de l'impulsion *START* qui déclenche le fonctionnement de l'unité. Cette dernière permet uniquement le calcul des deux premières adresses (00)h = 0 et (40)h = 64. En ce qui concerne le reste des adresses le calcul repose sur la valeur du signal d'entrée *CORTC* .

- Si *CORTC* = 1 au front montant de l'horloge, nous obtenons l'adresse suivante, comme le cas de (01)h = 1 et (41)h = 65 qui est suivi par le couple d'adresse (02)h = 2 et (42)h = 66…etc.
- Si *CORTC* = 0 au front montant de l'horloge, l'unité maintient la dernière adresse en attendant que *CORTC* passe à un pour continuer l'opération.

La séquence des adresses des 128 échantillons de départ est décomposée en deux séquences de 64 échantillons et cela est illustré à la figure (b) sur laquelle nous pouvons remarquer que l'unité a générée les couples {(0,40),….,(3F,7F)}h = {(0,64),…,(63,127)} en premier lieu avant de passer à deux blocs de 64 échantillons chacun {(0,20),…,(3F,5F)}h = {(0,32),…(63,95)} et {(40,60),…,(5F,7F)} = {(64,96),..(95,127)} et cela en employant la technique de l'entrelacement fréquentiel. L'opération est ensuite répétée sept fois jusqu'à aboutir aux transformées d'ordre un.

La fin des opérations est signalée par le passage du signal TC à un comme nous l'indique la figure (c).

III.3 L'unité papillon (Butterfly) [4, 13]

Ce module est dédié à l'implémentation de l'équation (11) vue au chapitre (I) et traduite par le graphe montré à la figure (2). Le module CORDIC (fig. 4) est le cœur de l'unité Papillon qui accepte les termes (P_{re}-Q_{re}) et (P_{im}-Q_{im}) comme étant les deux composantes du vecteur d'entrée et après avoir effectué une rotation de l'entrée d'un angle $k\theta$, il fournit le vecteur de sortie S.

Cependant, la sortie est réduite d'un facteur ξ_M = 1/0.607252935 = 1.646759 comme l'indique l'équation (18). Afin de restaurer la valeur propre du vecteur de sortie nous avons choisi de réduire la sortie R^* du même facteur ξ_M au lieu de multiplier le résultat S par la valeur 0.607252935 Cette opération est réalisée par le module SCALE (fig.21) qui garantie l'uniformité et le maintient d'une relation en phase entre les sorties, mais qui modifie l'amplitude du vecteur de sortie.

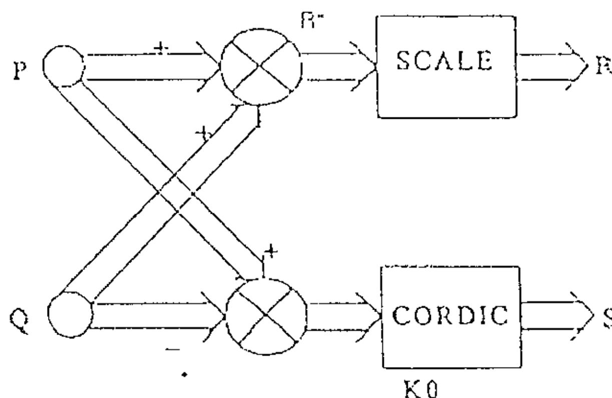


Figure 18. L'unité Papillon.

III.3.1 L'unité SCALE [4, 13]

Le bloc SCALE est chargé d'augmenter la valeur du vecteur (P_{re}+Q_{re}) et (P_{im}+Q_{im}) qui se trouve à son entrée d'un facteur ξ_M = 1.646759. Cette dernière valeur peut être écrite comme suit :

$\xi_M = 1 + 2^{-1}(1 + 2^{-2}(1 + 2^{-3}(1 + 2^{-2}(1 + 2^{-1}(1 + 2^{-3})))))$. ....(12).

Nous remarquons que ξ_M n'est autre qu'une somme de multiplication par puissance de moins deux. Sachant que la multiplication par puissance de moins deux peut être obtenue par des registres décaleurs à droite, et la somme des termes par un simple additionneur.

Après développement de l'équation (12), ξ_M s'écrit :

$\xi_M = 1 + 2^{-1} + 2^{-3} + 2^{-6} + 2^{-8} + 2^{-9} + 2^{-12}$.

Pour réaliser ce calcul nous faisons appel à deux décaleurs, l'un décale d'un bit et l'autre de deux bits. Cette technique nous permet de calculer le terme actuel en employant le terme précédent, afin d'éviter de reprendre les calculs depuis le début.

Le fonctionnement des registres décaleurs est contrôlé par un bus de deux bits D[0 :1] (fig.21), la valeur véhiculée sur ce dernier dépend du terme à calculer. Le tableau suivant présente les valeurs possibles véhiculées sur le bus D.

Terme à calculer	D_1	D_0	Nombre de bit à décaler.	Commentaire.
BS*1	0	0	0	Pas de décalage.
$BS*2^{-1}$	0	1	1 bit	Valider le décaleur d'un bit.
$BS*2^{-3}$	1	0	2 bits	Valider le décaleur de deux bits.
$BS*2^{-6}$	1	1	3 bits	Valider le deux décaleurs en même temps.
$BS*2^{-8}$	1	0	2 bits	Valider le décaleur de deux bits.
$BS*2^{-9}$	0	1	1 bit	Valider le décaleur d'un bit.
$BS*2^{-12}$	1	1	3 bits	Valider les deux décaleurs en même temps.

Tableau 1. Fonctionnement des décaleurs.

III.3.1.a Architecture fonctionnelle

Le schéma général de cette unité est constitué des bloc suivants :

- Le bloc SH16_1 : est un registre qui décale la valeur se trouvant à son entrée d'un bit.
- Le bloc SH16_2 : est semblable au bloc précédent sauf qu'il décale de deux bits.
- Le bloc CO et le bloc STATUS1 : représentent deux processus qui travaillent d'une manière concurrente. Leur tâche est de contrôler les deux décaleurs en leur fournissant les valeurs de D ainsi que de valider l'entrée du MUXREG16 (fig.21).

Le bloc CO est sensible au front montant du signal SLS en plus du niveau haut du signal STATS engendré par le bloc STATUS1, dès la réalisation de cette condition le bloc CO positionne la donnée sur le bus et fait passer STC à un afin de valider la donnée.

Le bloc STATUS1 est également sensible au front montant du signal SLS en plus du niveau haut du signal STC provenant du bloc CO dès lors la sortie STATS est mise à zéro.

- Le MUXREG16 : est un registre multiplexeur de 16 bits.

- Le bloc DATR16 : Est un registre buffer de 16 bits qui joue le rôle d'un accumulateur.

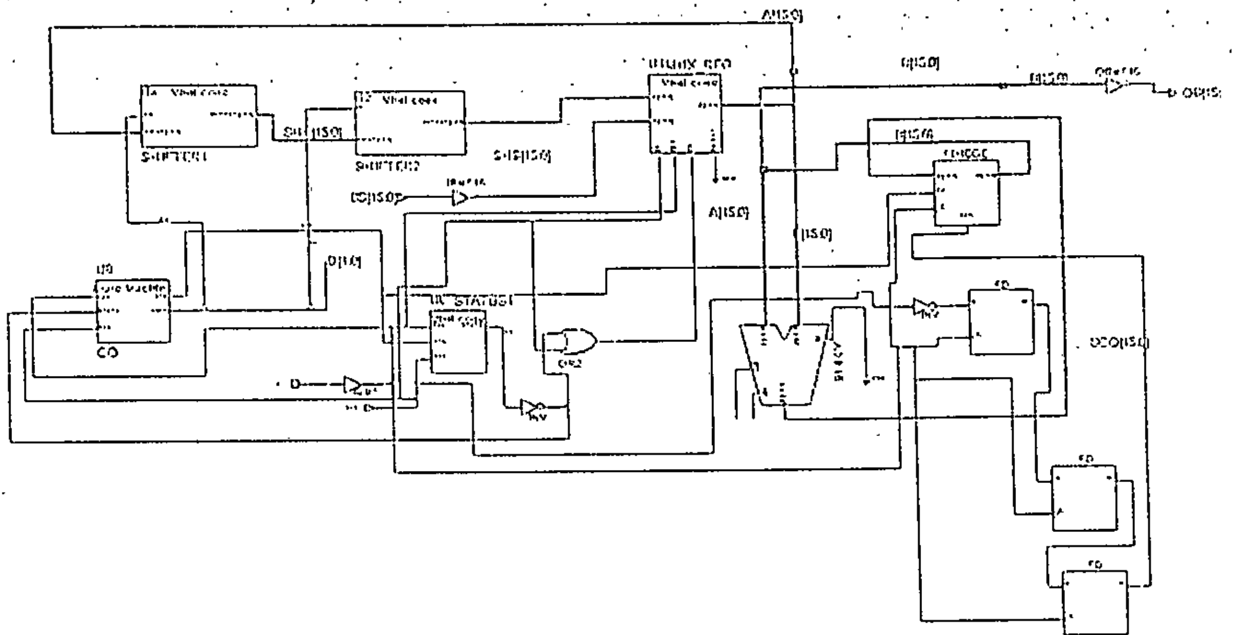


Figure 19. Le bloc SCALE.

III.3.1.b Simulation fonctionnelle et interprétation des résultats

Pour tester notre bloc nous avons attribuer au bus d'entrée BS la valeur suivante : *D6E7*, dont nous avons suivis l'évolution à travers notre circuit.

La simulation obtenue a la forme suivante :

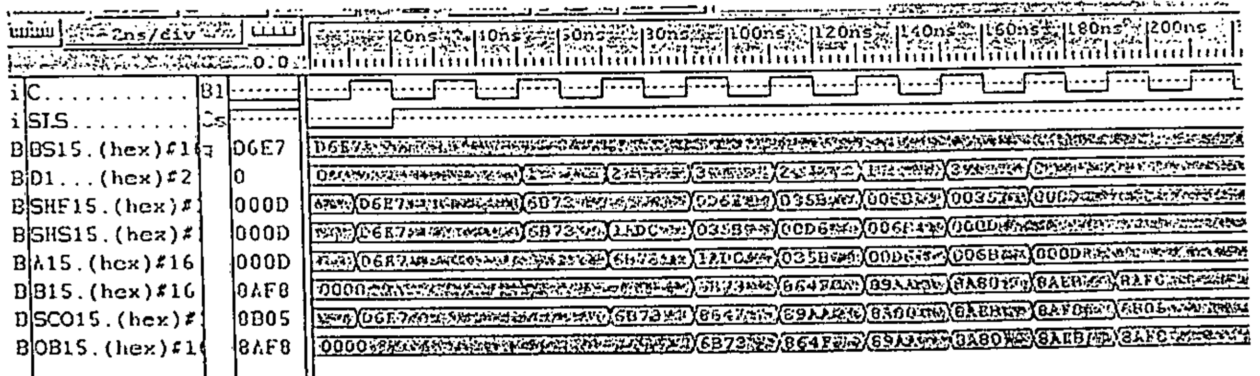


Figure 20. Simulation fonctionnelle du bloc SCALE.

III.3.2 L'unité CORDIC

III.3.2.a Résolution d'opérations mathématiques par le CORDIC

La série d'équations(18) permet selon le mode utilisé, le calcul des opérations mathématiques suivante : *SINUS*, *COSINUS*,*RACINE CARREE* et *ARCTANGENTE*. Ces deux modes sont :

- Le mode rotation [4, 13]

Le mode rotation calcule, simultanément, à partir d'un angle initial donné par l'utilisateur, le *SINUS* et le *COSINUS* de ce dernier. Nous initialisons respectivement les variables X_0, Y_0, E_0 à 0.6073, 0 et α où α est l'angle dont le *SINUS* et le *COSINUS* sont recherchés. L'angle pivote jusqu'à ce qu'il ait atteint la valeur 0. Les valeurs Xi et Yi résultantes expriment alors le *SINUS* et le *COSINUS*. Un test à chaque itération sur l'angle élémentaire E_i détermine le signe de δ_i.

Notons que nous avons choisi ici un angle de 25.31 degré et un nombre d'itération égale à 15.

- le mode vecteur [4, 13]

Par opposition, le mode vecteur calcule simultanément, à partir des coordonnées X et Y données par l'utilisateur, les fonctions $Arc\tan(\frac{y}{x})$ et $\zeta_M\sqrt{x^2+y^2}$. Nous initialisons respectivement les variables X_0, Y_0, E_0 à X, Y et 0. Cette fois l'angle pivote jusqu'à ce que la variable Y atteigne 0. Les valeurs X_i et E_i résultantes expriment alors la *RACINE CARREE* et l'*ARCTANGENTE* respectivement. Notons que pour obtenir la valeur finale de la *RACINE CARREE*, nous devons diviser le résultat par E_n. Un test sur Y_i détermine pour chaque itération le signe δ_i. Nous avons choisi les valeurs Xo=4, Yo=4 et un nombre d'itérations égale à 15.

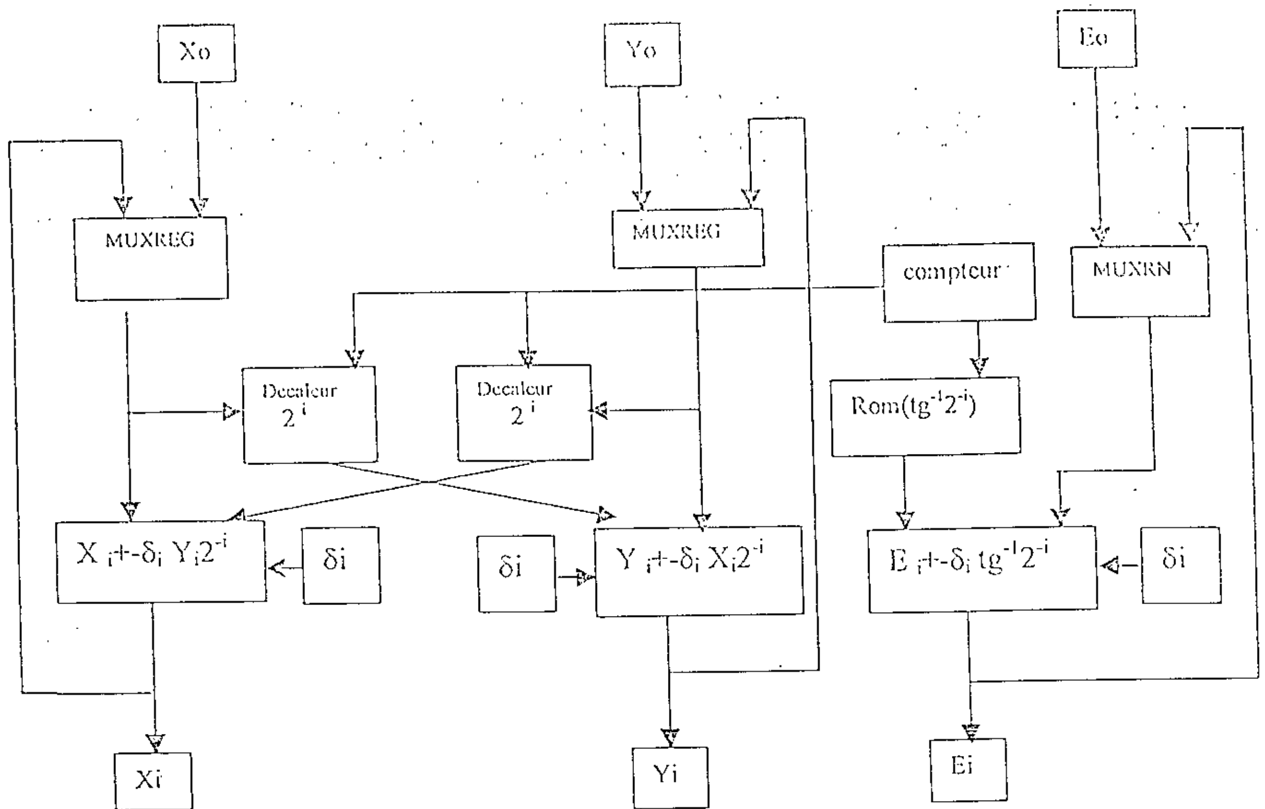


Figure 21 .Schema fonctionnel du cordic.

III.3.2.b Simulations et résultats

Mode rotation :MAG='0'

Spécifications

Le nombre d'itérations maximal est de 15(4 bits). Nous suggérons d'utiliser une précision fractionnaire de 14 bits pour les coordonnées X et Y. Ainsi les registres X et Y devraient avoir la dimension suivante[4, 13] :

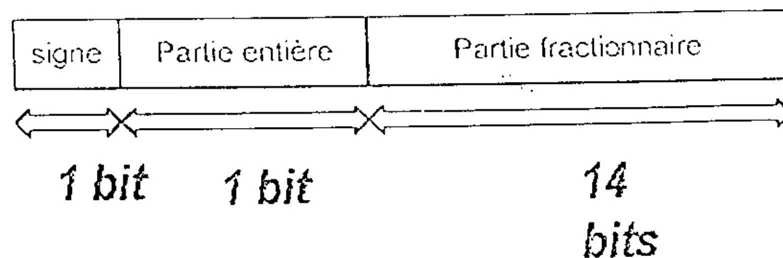
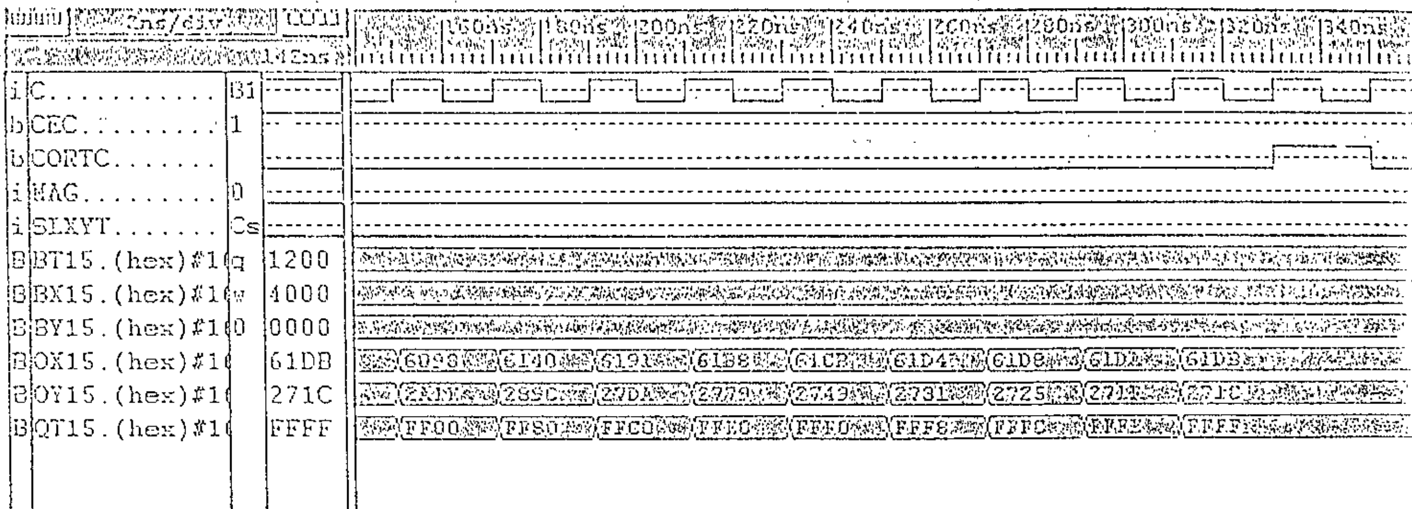


Figure 22. Format de la donnée dans le mode rotation .



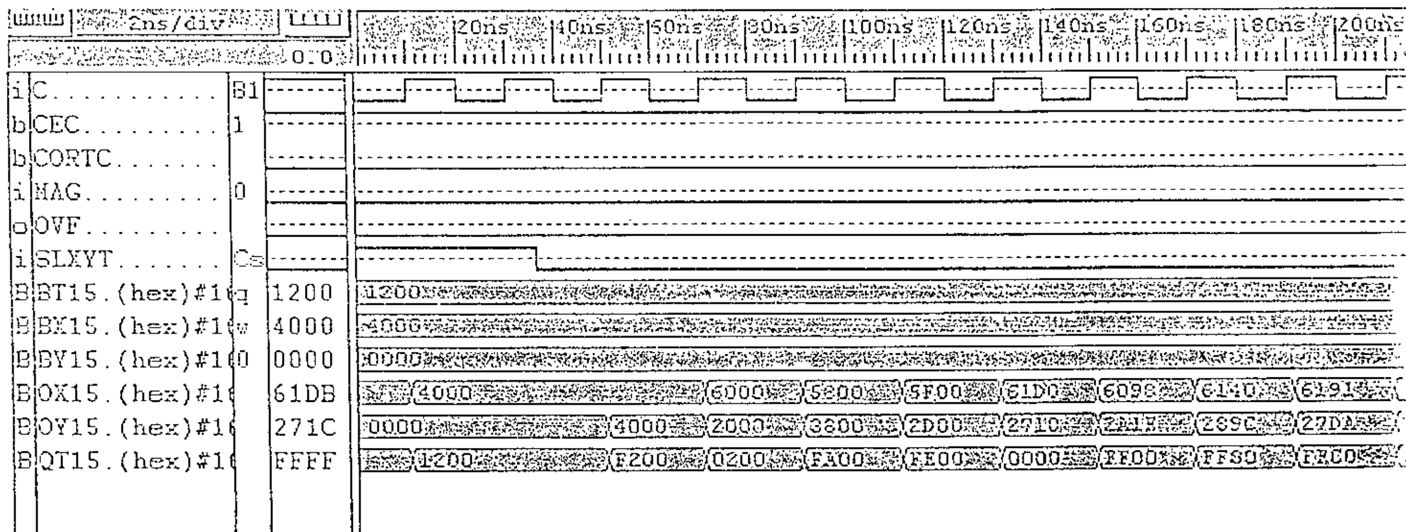


Figure 23. Chronogramme du CORDIC dans le mode rotation

Nous avons choisi de générer le *cosinus* et le *sinus* de l'angle (π/8+π/32)=25,31 degrés nous initialisons à 1,0 et 25.21 les entrées *X*, *Y* et *E*, les sorties *OX*, *OY* représentent respectivement les *COS* et *SIN*, sachant que *QT* doit s'annuler.

Nous choisissons de prendre un incrément de départ du signal *SLXYT* de 35ns.

Le gain de l'algorithme est de 1,647 donc toutes les sorties doivent être réduite du même facteur.

- La valeur 61DB (cos25.21)=0.91
- La valeur 271C (sin25.21)=0.38

Nous remarquons que l'angle est négligeable FFFF.

(i)	X		Y		E	
	Hexa	*Décimal*	*Hexa*	*Décimal*	*Hexa*	*Décimal*
0	4000	1	0000	0	1200	25.312°
1	4000	1	4000	1	F200	-42.187°
2	6000	0.911	2000	0.303	0200	2.812°
3	5800	0.835	3800	0.531	FA00	-8.440°
4	5F00	0.901	2D00	0.426	FE00	-2.812°
5	61D0	0.928	2710	0.370	0000	0
6	6098	0.916	2A1E	0.372	FF00	-1.406°
7	6140	0.922	289C	0.372	FF80	-0.703°
8	6191	0.925	27DA	0.372	FFCO	-0.175°
9	61B8	0.927	2779	0.372	FFE0	-0.087°
10	61CB	0.927	2749	0.372	FFF0	-0.043°
11	61D4	0.928	2731	0.371	FFF8	-0.021°
12	61D8	0.928	2725	0.371	FFFC	-0.010°
13	61DA	0.928	271F	0.371	FFFE	-0.005°
14	61DB	0.928	271C	0.371	FFFF	-0.002°

Tableau 3. Résultats de la simulation en utilisant le mode rotation.

Mode vectoriel :MAG ='1'

Spécifications

Le nombre d'itérations maximal est de 15(4 bits). Nous suggérons d'utiliser une précision fractionnaire de 11 bits pour les coordonnées *X* et *Y*. Ainsi les registres *X* et *Y* devraient avoir la dimension suivante[4, 13] :

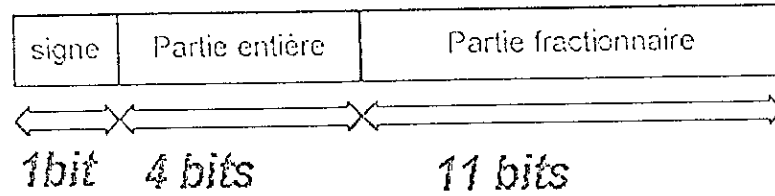


Figure 24. Format des données X etY dans le mode vecteur.

Le format de l'angle E est le suivant:

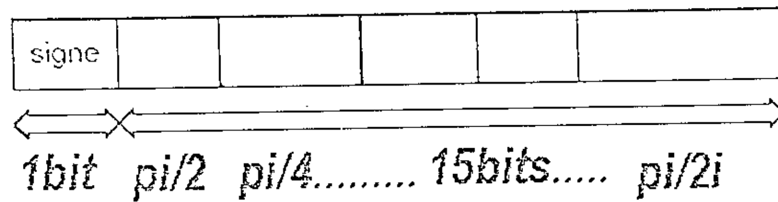
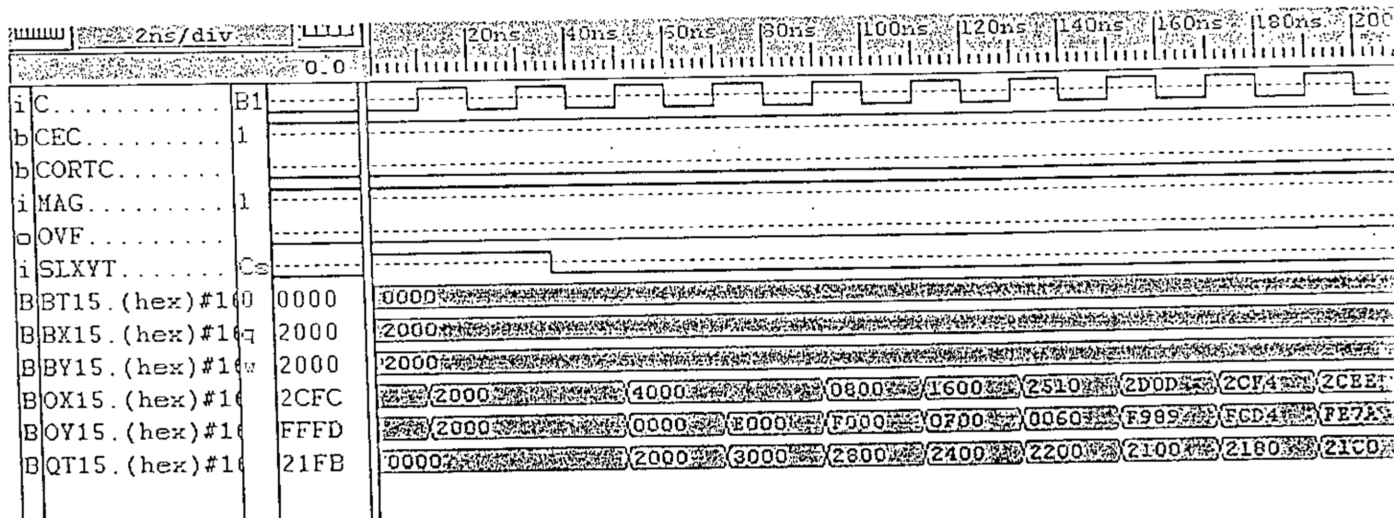


Figure 25. Format de la donnée E .



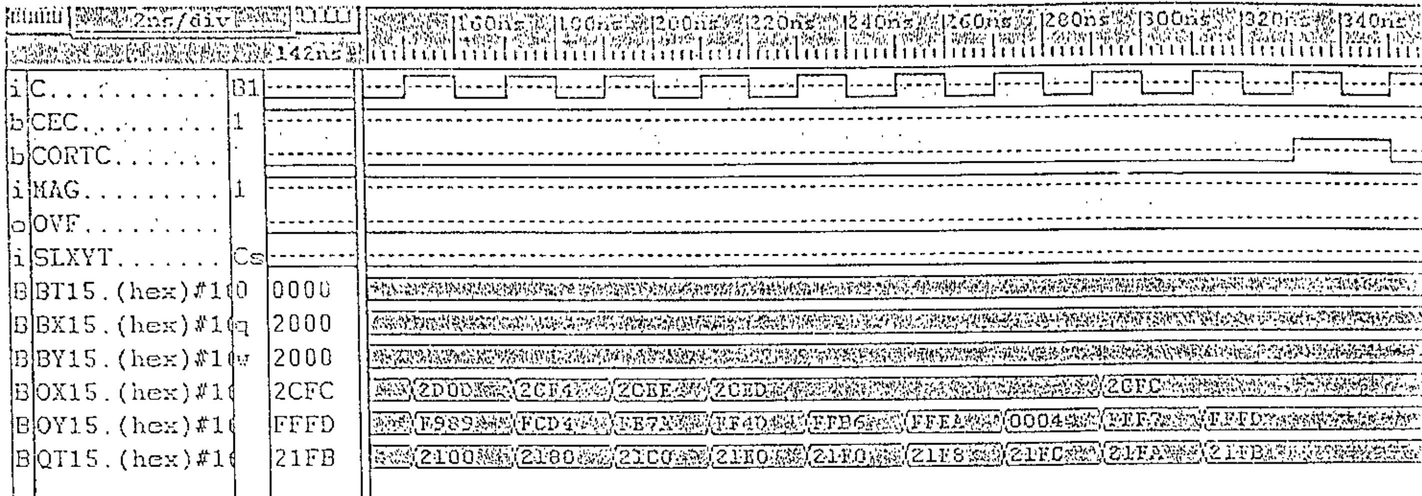


Figure 26. *Chronogramme du CORDIC dans le mode vecteur*

I	X		Y		E	
	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
0	2000	4	2000	4	0000	0
1	4000	8	0000	0	2000	45°
2	4000	8	E000	-3.999	3000	67.5°
3	0800	1	F000	-2.990	2800	56.25°
4	1600	2.75	0F00	1.875	2400	50.625°
5	2510	4.632	0060	0.046	2200	47.812°
6	2D0D	5.630	F989	-0.808	2100	46.406°
7	2CF4	5.619	FCD4	-0.395	2180	47.109°
8	2CEE	5.616	FE7A	-0.190	21C0	47.285°
9	2CED	5.615	FF4D	-0.087	21E0	47.417°
10	2CED	5.615	FFB6	-0.043	21F0	47.417°
11	2CED	5.615	FFEA	-0.01	21F8	47.417°
12	2CED	5.615	0004	0.001	21FC	47.417°
13	2CFC	5.623	FFF7	-0.002	21FA	47.417°
14	2CFC	5.623	FFFD	-0.001	21FB	47.417°

Tableau 4. Résultats de la simulation en utilisant le mode vecteur.

Nous remarquons que la sortie y tend vers la valeur nulle, l'angle tend vers 45° avec une marge d'erreur, ce qui correspond à *arctangente* (4/4), et la sortie X tend quant à elle vers 5.623 ce qui correspond à $\sqrt{X^2+Y^2}$.

III.3.2.c Architecture fonctionnelle :

Le schéma du *CORDIC* contient trois additionneurs/soustracteurs et deux décaleurs variables *VSH16* commandés par un compteur *CB4CE*, à chaque itération il décale la valeur entrée, ce décalage fait office de multiplication de cette valeur par 2^{-i} où (i) représente le nombre d'itérations.

Le registre multiplexeur *MUXREGISTER16* charge en premier lieu la valeur initiale puis la valeur interne intermédiaire à chaque itération et cela grâce au signal *SLXYT*.

Le bloc *THETA* réagit en *ROM* qui stocke les valeurs de l'angle $\tan^{-1}(2^{-i})$ utilisées pour la comparaison avec l'angle cible approché.

Le compteur marque la fin de calcul du *CORDIC* par un signal *CORTC* préparant ainsi le début d'une nouvelle phase de calcul .

La ligne de contrôle *MAG* décide du mode d'opération du *CORDIC*, soit en rotation ou en vecteur, commandant ainsi les deux multiplexeurs, un pour la commande du sens de rotation et l'autre pour la commande du signe des sorties *X* et *Y*.

En mode rotation le sens de rotation est déterminé par l'erreur instantanée de l'angle calculé qui est la différence entre l'angle de départ et l'angle précèdent : $E_{i+1}=E_i-\delta_i \tan^{-1}(2^{-i})$. [4, 13]

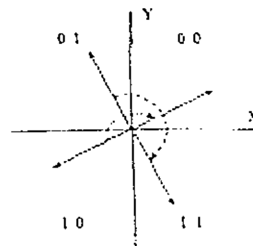


***Figure 27**. Le maintient de l'angle dans le 1er et $4^{ème}$ quadrant*

L'angle de départ est chargé dans un registre multiplexeur de structure spécifique *mux_registre_16N*, puisqu'il copie le $14^{ème}$ bit de l'angle de départ dans le $15^{ème}$ bit (MSB) figure(27), sachant que le MSB est le bit de signe en complément à deux, le quatorzième représente l'angle π/2 et le treizième l'angle π/4 et ainsi de suite.

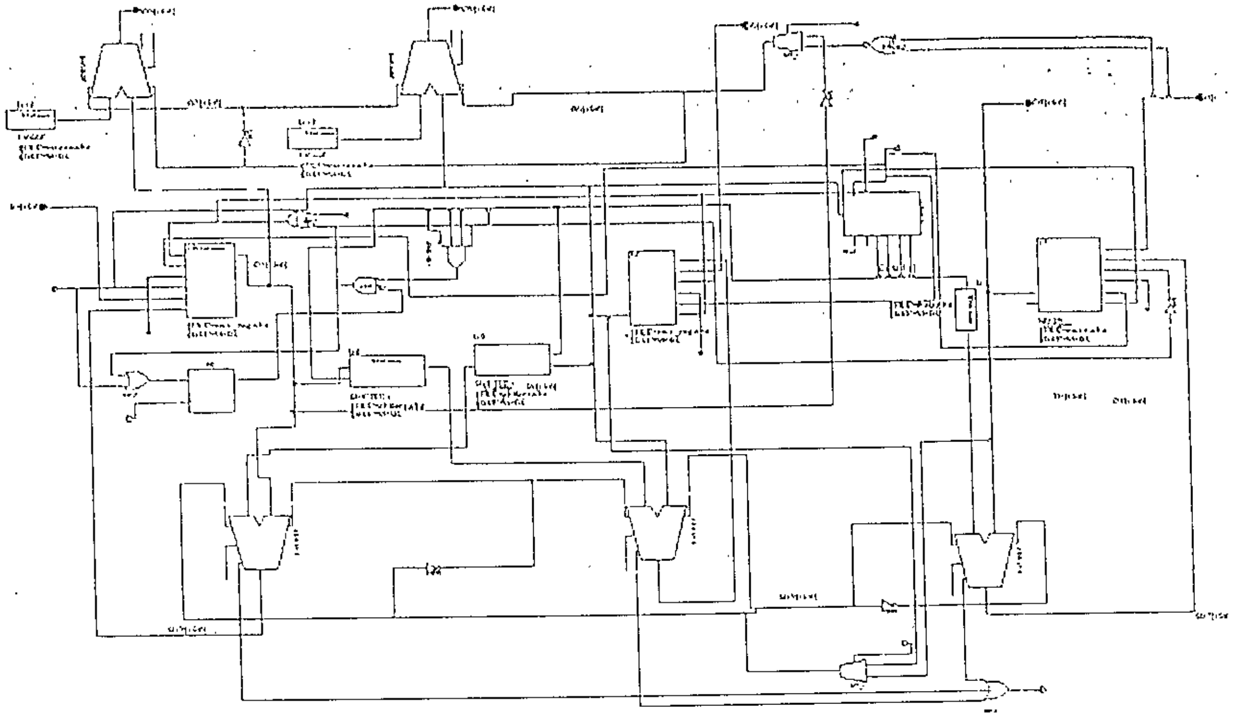


Figure 28. Le bloc CORDIC.

Exemple : supposons un angle $5\pi/8$ de départ sur 6 bits :*010100* ce qui donne *110100* en complément à 2 on aura *001011+1=001100*, donc l'angle résultant de la manipulation est $-3\pi/8$ ce qui représente une diminution d'un angle π de l'angle de départ et donc nous sommes arrivé à remettre le point dans le 1er et 4ème quadrant du plan puisque le *CORDIC* ne travaille que dans ces deux cadrans donc dans l'intervalle $[-\pi/2, \pi/2]$ puis compenser cette diminution d'un l'angle π, Le signe des sorties *X* et *Y* dépend de la position de l'angle dans le plan s'ils sont en dehors du 1er et 4ème cadran les sorties sont inversées en complément à deux, mais dans le mode vectoriel le sens de rotation dépend du signe de *Y* quant au signe des sorties *X* et *Y* il dépend du signe de *X*.

Le schéma global a été réalisé en schématique de XILINX FOUNDATION, du fait de la facilité de description de composant comme le compteur ou l'additionneur, ceci dit les blocs à fonctions spécifiques ont été réalisés en VHDL, prenons comme exemple les *décaleurs*, la rom *THETA* ou le *mux_register_16N*.

Pour plus de détails en ce qui concerne le fonctionnement de l'unité CORDIC veuillez consulter *l'Annexe C*.

III.4 L'unité de contrôle

Le processeur de la FFT inclus trois blocs mémoires *RAMI*, *RAMII* et *RAMY* chacune d'elles ayant une capacité de 128 mots de 16 bits. Après l'initialisation, les 128 échantillons délivrés par l'*ADC* sont chargés dans *RAMI*. L'emplacement du chargement est pointé par le compteur (*COUNT*) qui est chargé de générer les adresses d'une manière séquentielle jusqu'au 128ème échantillon; après la fin du balayage de la mémoire le bloc *COUNT* génère un signal *TL* indiquant la fin du chargement de *RAMI*.

La seconde phase qui représente la phase de calcul de la FFT est réalisée sur plusieurs étapes:L'unité de génération d'adresses génère le premier couple d'adresses qui va pointer le premier couple de données dans la mémoire sur lequel sera effectué le traitement.

Le traitement effectué sur le couple de données se fait comme suit:

- Les deux valeurs sont soustraite l'une à l'autre dans le bloc *SUMDIF* et sont envoyées au *CORDIC* puis elles sont sommées et sont envoyées à

SCALE, sachant que le bloc *SUMDIF* fait la différence puis la somme successivement.

- Le résultat (*R*) calculé par *SCALE* est rangé dans le mot mémoire adressé par la première adresse désignée par le bloc de génération d'adresses et le résultat (*S*) calculé par le *CORDIC* est aussi rangé dans un deuxième mot mémoire désigné aussi par l'unité de génération d'adresses.

  La partie imaginaire des résultats du *CORDIC* est stockée dans *RAMY*.

  Lorsque les 128 échantillons sont mémorisés, le contrôleur reçoit un signal TC de l'unité de génération d'adresses et envoie à son tour un signal au bloc DISP pour afficher les résultats finaux.

  Le bloc *SWITCH1* sert à sélectionner une de ses entrées sur ordre de l'unité de contrôle.

  En parallèle à chaque calcul du module *CORDIC* et du module *SCALE*, les valeurs de la deuxième série d'échantillons sont introduites dans la *RAMII*.

Le contrôleur est le cerveau de toute l'architecture, il est chargé de gérer le fonctionnement de l'ensemble des modules en suivant une séquence bien déterminée. La conception de cette unité fait appel au concept de machine d'états finis (*FSM*).

Pour éclairer son fonctionnement nous donnons les résultats de sa simulation ci-dessous.

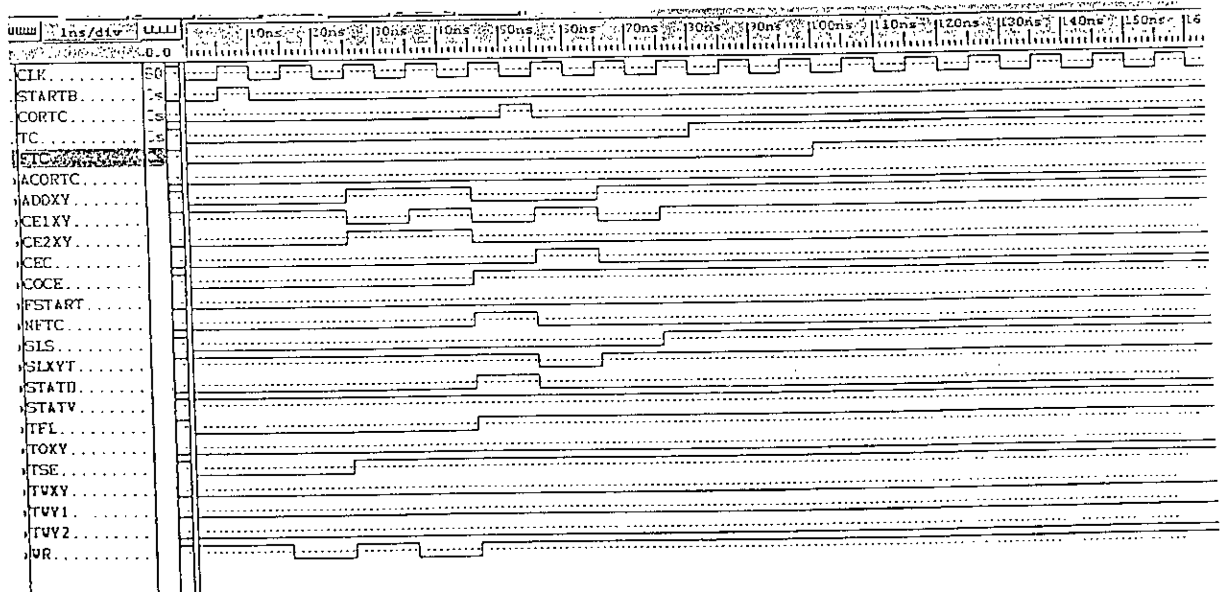


Figure 29. Chronogramme de simulation du contrôleur.

Pour plus de détails en ce qui concerne le fonctionnement de l'unité de contrôle veuillez consulter *l'Annexe D.*

Après la validation des différents blocs nous les assemblons et nous obtenons ainsi le schéma global représenté dans la figure suivante:

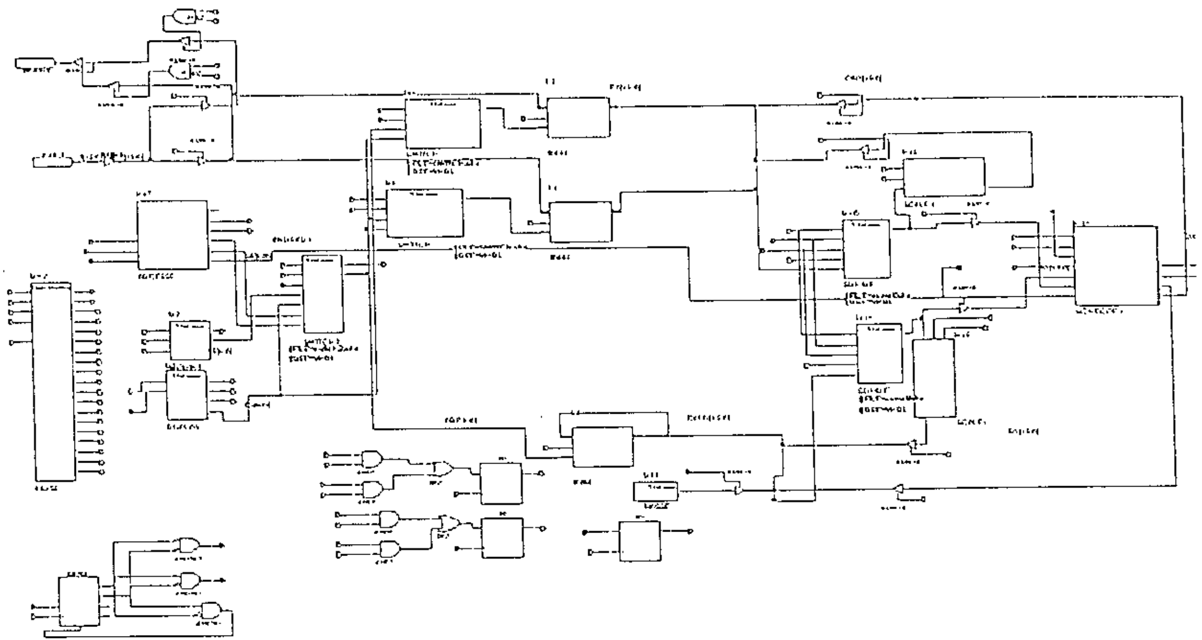


Figure 30. *Schéma global du processeur FFT*

III.5 - Conclusion

La validation des "designs" est un élément primordial du travail de conception puisqu'elle permet de diminuer les risques et les coûts en permettant une détection précoce des anomalies.

La démarche de conception, à ce stade, se décompose ainsi:

- Conception structurée, modulaire et claire.
- Simulation fonctionnelle.

Cette démarche nous a facilitée la création de notre design grâce à la méthode "*top-down*" ainsi que la validation de nos résultats à l'aide de l'approche "*bottom-up*". Après l'achèvement de cette phase, nous entamons dans le chapitre suivant l'étape de la synthèse.

A défaut de simuler et d'implémenter le schéma global en raison de la limitation des ressources du logiciel XILINX FOUNDATION nous avons néanmoins entrepris d'implémenter les trois blocs les plus importants sur des circuits différents :

- Le bloc de génération d'adresses sur le 4003.
- Le bloc SCALE sur le 4003.
- Le bloc CORDIC sur le 4006.

Passant par toutes les étapes d'implémentation qui sont :

- La simulation fonctionnelle.
- La synthèse.
- Le placement-routage.
- La simulation temporelle.

Chapitre 4

Synthèse, Placement-Routage et Simulation temporelle

IV.1 SYNTHÈSE ET PLACEMENT-ROUTAGE.....................63

IV.1.1 INTRODUCTION 63

IV.1.2 SYNTHÈSE : LE CONCEPT 63

IV.1.3 CRITÈRES DE PERFORMANCES 64

IV.1.3.a Puissance de calcul..64

IV.1.4 ETAPES DE LA SYNTHÈSE 67

IV.1.5 SYNTHÈSE ET PLACEMENT-ROUTAGE DU BLOC DE GÉNÉRATION D'ADRESSES 68

IV.1.6 LA SYNTHÈSE DU BLOC SCALE 69

IV.1.7 LA SYNTHESE DU BLOC CORDIC 72

IV.2 SIMULATION TEMPORELLE 73

IV.2.1 INTRODUCTION 73

IV.2.2 SIMULATION TEMPORELLE DU BLOC DE GÉNÉRATION D'ADRESSE 73

IV.2.3 SIMULATION TEMPORELLE DU BLOC SCALE............74

IV.2.4 SIMULATION TEMPORELLE DU BLOC CORDIC.........75

IV.3 CONCLUSION 76

IV - Synthèse, palcement-routage et simulation temporelle.

IV.1 Synthèse et placement-routage. [8, 12, 14]

IV.1.1 Introduction

Cette phase est prise en charge pour l'essentiel par les outils de C.A.O. Il s'agit de la phase de synthèse logique suivie du placement/routage du circuit à réaliser. Nous nous contenterons ici de survoler les aspects principaux pour donner les grandes idées propres à cette phase. L'outil principalement utilisé n'est plus le simulateur VHDL, mais le synthétiseur logique, qui à partir d'une description VHDL de la conception d'un système, génère une « netlist » logique du circuit correspondant.

Il s'agit d'un schéma logique instanciant des éléments de la bibliothèque technologique utilisée. C'est ce schéma qui est ensuite placé et routé pour obtenir le circuit final. L'essentiel du travail dans cette phase consiste à optimiser les propriétés du circuit généré en termes de taille et de caractéristiques temporelles, comme la fréquence maximale de l'horloge.

Cette optimisation est entièrement propre à l'outil utilisé. Le respect de ces contraintes temporelles ou de taille peut ne pas être obtenue, conduisant à modifier la conception ou même parfois les spécifications.

IV.1.2 Synthèse : le concept

« Synthétiser » signifie traduire une description textuelle d'une fonction en une interconnexion de modules physiques. Dans le cas qui nous intéresse, ces modules physiques correspondent à des cellules d'une librairie définie soit par un fabricant de FPGA, soit par un fondeur.

Ces cellules correspondent aux fonctions élémentaires disponibles avec la technologie utilisée.

Généralement, la librairie comporte un certain nombre de fonction logiques telles que des OU, ET, OU exclusifs, ainsi que différents types de flip-flops. Le VHDL fourni par le designer utilise un niveau de description plus élevé, où les signaux sont des bus, les mémoires décrites avec des évènements (*events*) et les opérations effectuées plus proches des opérations arithmétique que des opérations logiques. Ce niveau de description est d'ailleurs souvent appelé RTL (*Registr Transfer Level*) car on décrit la manière dont les données sont transférées et modifiées d'un registre à un autre.

IV.1.3 Critères de performances

Outre la technologie de programmation, capacité et vitesse sont les maîtres mots pour comparer deux circuits (PLDs, CPLDs, FPGAs,...). Mais quelle capacité et quelle vitesse ?

IV.1.3.a Puissance de calcul

Les premiers chiffres accessibles concernent les nombres d'opérateurs utilisables.

Nombre de portes équivalentes

Le nombre de portes est sans doute l'argument le plus utilisé dans les effets d'annonce. En 2000 la barrière des 250 000 portes est largement franchie. Plus délicate est l'estimation du nombre de portes qui seront inutilisées dans une application, donc le nombre réellement utile de portes.

Nombre de cellules

Le nombre de cellules est un chiffre plus facilement interprétable : le constructeur du circuit a optimisé son architecture, pour rendre chaque cellule capable de traiter à peu près tout calcul dont la complexité est en relation avec le nombre de bascules qu'elle contient (une ou deux suivant les architectures). Trois repères chiffrés : un 22V10 contient 10 bascules, la famille des CPLDs va de 32 bascules à quelques centaines et celle des FPGAs s'étend d'une centaine à quelques milliers.

Nombre d'entrées/sorties

Le nombre de ports de communication entre l'intérieur et l'extérieur d'un circuit peut varier dans un rapport de deux, pour la même architecture interne, en fonction du boîtier choisi. Les chiffres vont de quelques dizaines à quelques centaines de broches d'entrées/sorties.

Capacité mémoire

Les FPGAs à SRAM contiennent des mémoires pour stocker leur configuration. La plupart des familles récentes offrent à l'utilisateur la possibilité d'utiliser certaines de ces mémoires en tant que telles. Par exemple, la famille 4000 de XILINX permet d'utiliser les mémoires de configuration d'une cellule pour stocker 32 bits de données ; la cellule correspondante n'est évidemment plus disponible comme opérateur logique. Les capacités de mémorisation atteignent quelques dizaines de Kilobits.

Routabilité

Placement et routage sont intimement liés, et le souhait évident de l'utilisateur est que ces opérations soient aussi automatiques que possible. Le critère premier de routabilité est l'indépendance entre la fonction et le brochage. Certains circuits garantissent une routabilité complète : toute fonction intégrable dans le circuit pourra être modifié sans modification du câblage externe.

Le routage influe sur les performances dynamiques de la fonction finale. La politique généralement adoptée est de prévoir des interconnexions hiérarchisées : les cellules sont regroupées en grappes entre elles. Les interconnexions locales n'ont que peu d'influence sur les temps de calcul, contrairement au interconnexions distantes dont l'effet est notable.

A priori c'est au placeur–routeur que revient la gestion de ces interconnexions; à condition que le programmeur ne lui complique pas inutilement la tâche. Un optimiseur a toujours du mal à découper des blocs de grandes tailles, il lui est beaucoup plus simple de placer des objets de petites dimensions. En VHDL cela s'appelle construction hiérarchique ; un ensemble complexe doit être conçu comme l'assemblage d'unités de conceptions aussi simples que possibles.

Modèle général de détermination de fmax

Le modèle général de détermination de la fréquence maximum d'un opérateur séquentiel prend en compte les retards dans les circuits et les règles concernant les instants de changement des entrées vis-à-vis des fronts actifs de l'horloge. La figure (31) définit les temps les plus importants : t_{pi} pour des temps de propagation et t_{SU} pour le temps de pré-positionnement d'une bascule.

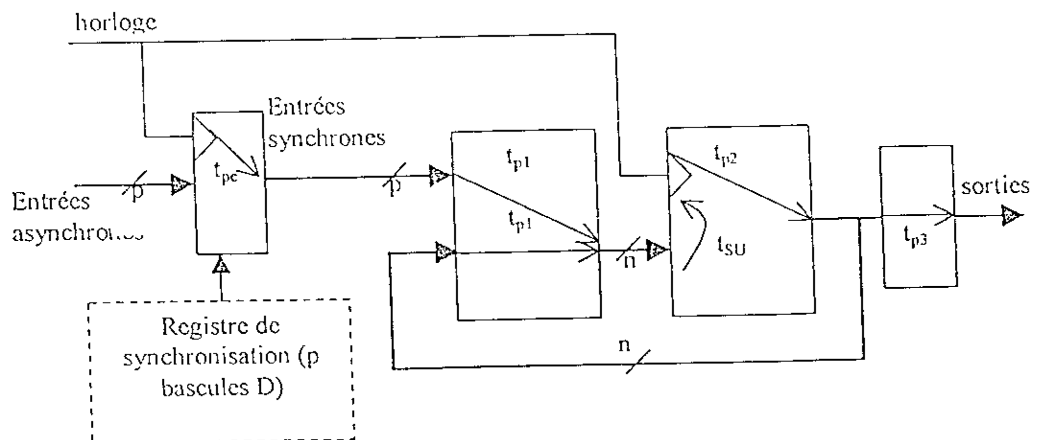


Figure 31. Modèle de calcul de la fréquence maximum.

La fréquence maximum de fonctionnement interne est donnée par :

$$Fint = 1/(t_{p2} + t_{p1} + t_{SU})$$

Pour le calcul de la fréquence maximum externe, il convient de rajouter, dans la forme précédente, le temps de propagation des cellules de sortie:

$$Fext = 1/(t_{p3} + t_{p2} + t_{p1} + t_{SU})$$

Dans un FPGA le routeur analyse le schéma généré et en déduit les différents temps de propagation, à partir d'un modèle des cellules élémentaires du circuit.

Dans le cas des PLDs simple et de beaucoup (pas tous) des CPLDs, les notices fournissent directement les valeurs des fréquences maximum et/ou des temps de retard et de pré-positionnement entre les signaux appliqués aux broches du circuit et les fronts de l'horloge.

IV.1.4 Etapes de la synthèse

La synthèse se déroule sur plusieurs phases et à chaque phase XILINX FOUNDATION rédige un rapport qui comporte les informations suivantes :

- Le rapport de conversion(Translation Report)

Si au moment de la conversion de la netlist en un format interne utilisé par l'outil d'implémentation une erreur est détecté, elle sera listée à ce niveau.

- Le rapport de mapping (Map Report)

Ici, vous serez informé de toutes les optimisations qui ont été réalisées sur la "netlist". Par exemple des portes logiques peuvent être ajoutées ou supprimées sans changer pour autant la fonction du circuit.

- Le rapport de placement-routage (Place & Route Report)

Ce rapport contient les options sélectionnées afin de réaliser le placement-routage. Il mentionne les erreurs et le temps total nécessaire pour réaliser les différentes phases de placement-routage.

- Le rapport des pins (Pad Report)

Il décrit les interconnexions entre les entrées-sorties et les pins.

- Le rapport de retard asynchrone (Asynchrounous Delay Report)

Dans ce rapport sont listés les retards de propagation de chaque signal routé.

- Le rapport temporel Post Layout(Post Layout Timing Report)

Ce rapport mentionne si un des chemins généré par la logique de placement et routage a violé une contrainte temporelle.

▸▸ Le rapport de génération de bits(Bitgen Report)

Ce rapport mentionne toutes les opérations qui ont été mise en œuvre lors de la génération du fichier binaire.

A présent, passons à l'analyse de la synthèse des blocs vus au chapitre III.

IV.1.5 Synthèse et placement-routage du bloc de génération d'adresses

Les figures suivantes présentent les résultats de placement, de routage et la forme finale du circuit cible après achèvement de la synthèse sur le circuit XC4003E-1-PC84 FPGA.

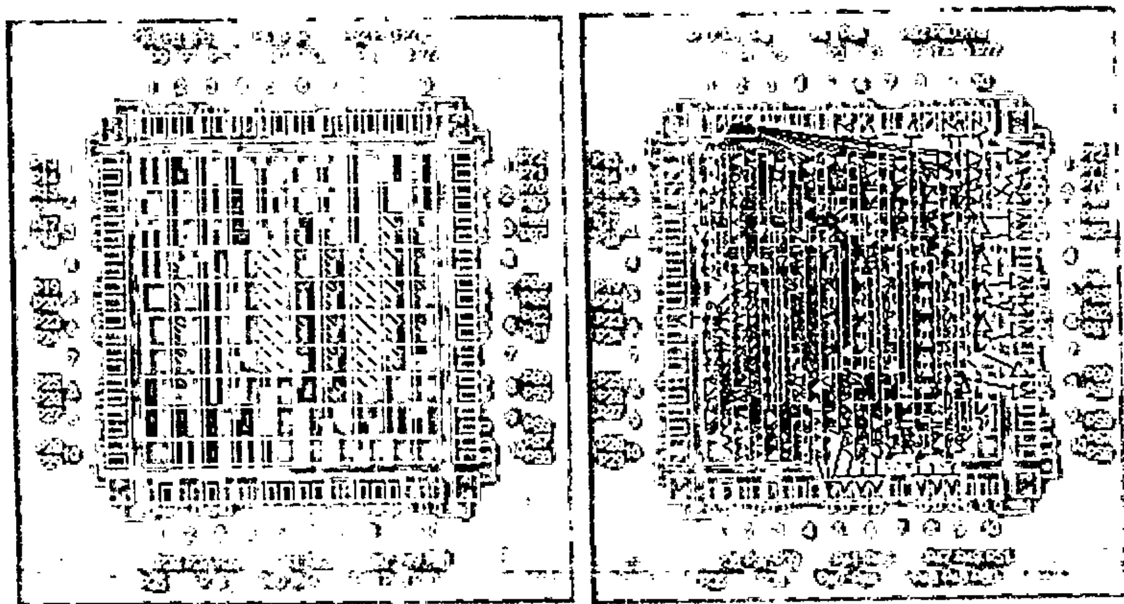


Figure 32. Le bloc de génération d'adresses après la phase de placement et de routage.

	Nombre utilisé	*Nombre disponible*	*Pourcentage (%)*
CLBs	97	100	97%
Bascules des CLBs	80	200	40%
IBOs	36	61	59%
4 input LUTS	178	200	89%
3 input LUTS	33	100	33%
Nombre totale de portes logiques équivalentes : 1990			
Nombre de portes JTAG additionnelles pour les IOBs : 1728			
Chemins (Paths) : 3035			
Nœuds (Nets) : 234			
Connexions : 685			

Période minimale : 29.920 ns
Fréquence maximale : 33,422 MHz.
Retard maximum engendré par un nœud :20.194 ns.

Tableau 5 . Les principaux résultats des rapports issus de la synthèse du bloc de génération d'adresses

En examinant les données du tableau (5) et la distribution des IOBs et CLBs sur notre circuit, nous pouvons en déduire que les ressources de notre circuit sont bien exploitées.

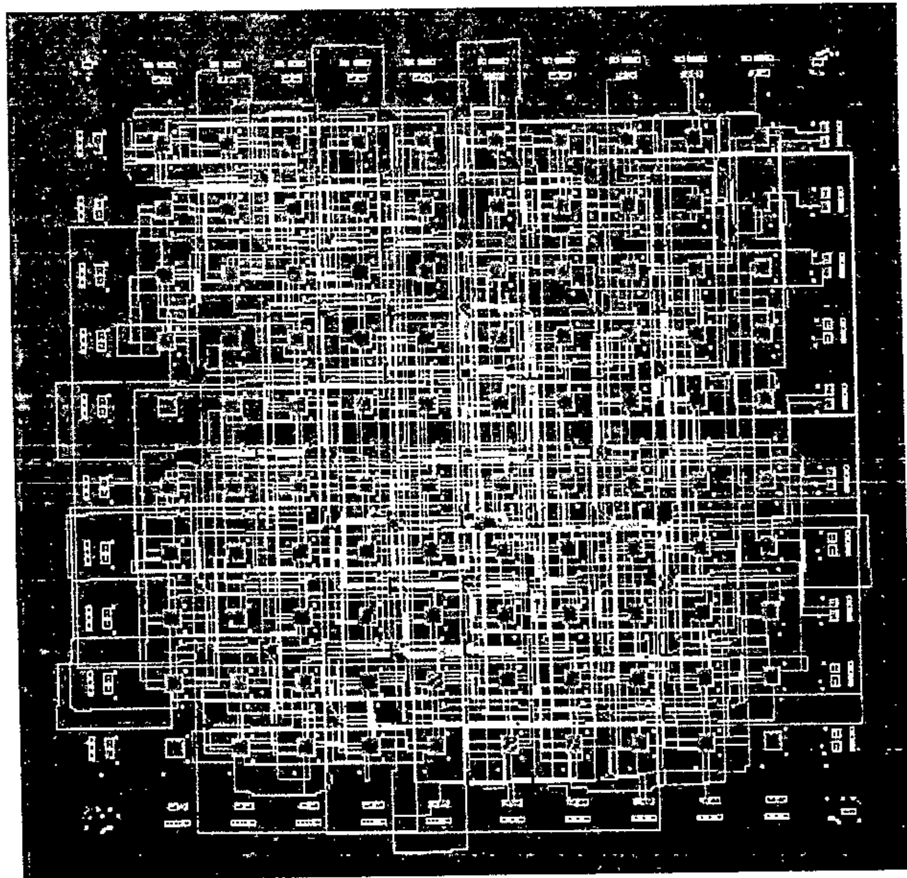


Figure 33*. Circuit représentant le bloc de génération d'adresses .*

IV.1.6 La synthèse du bloc SCALE

Le circuit cible utilisé est le XC4003E-1-PC84 FPGA.

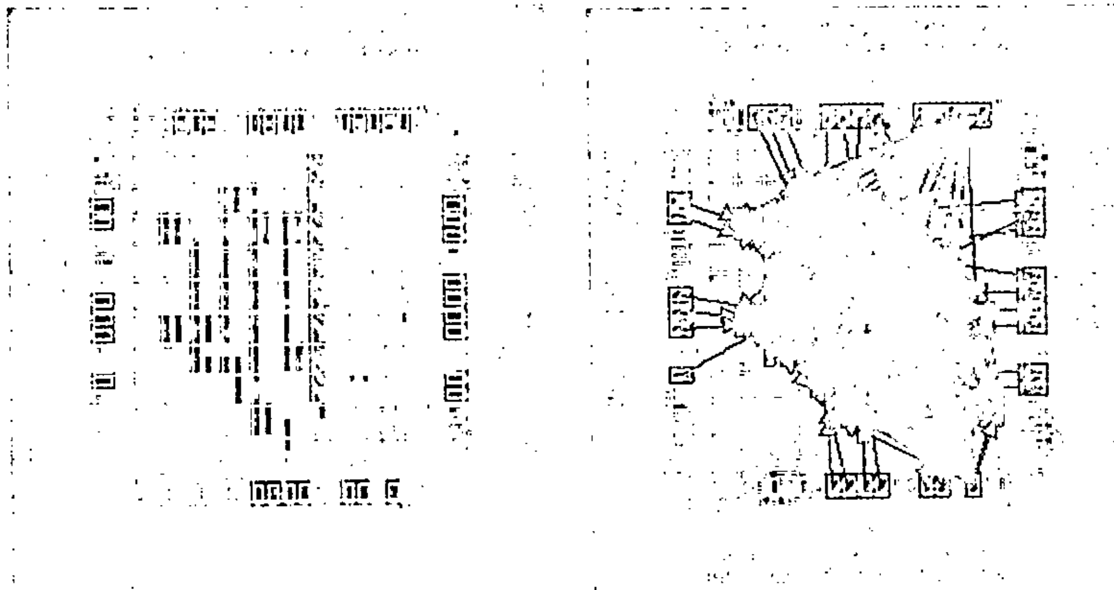


Figure 34. Le bloc SCALE après la phase de placement et routage

	Nombre utilisé	*Nombre disponible*	*Pourcentage (%)*
CLBs	54	100	54%
Bascules des CLBs	46	200	23%
IOBs	34	61	55%
4 input LUTS	99	200	49%
3 input LUTS	7	100	7%
Nombre totale de portes logiques équivalentes : 1002			
Nombre de portes JTAG additionnelles pour les IOBs : 1632			
Chemins (Paths) : 2870			
Nœuds (Nets) : 128			
Connexions : 356			
Période minimale : 25.460 ns			
Fréquence maximale : 39.268 MHz.			
Retard maximum engendré par un nœud :14.152 ns.			

Tableau 6 . Les principaux résultats des rapports issus de la synthèse du bloc SCALE

En analysant les résultats du tableau (6), nous constatons que notre circuit est moyennement exploité, néanmoins nous tenons à vous informer que nous avons utiliser le plus petit circuit disponible et pouvant supporter le bloc SCALE dans cette version de XILINX FOUNDATION.

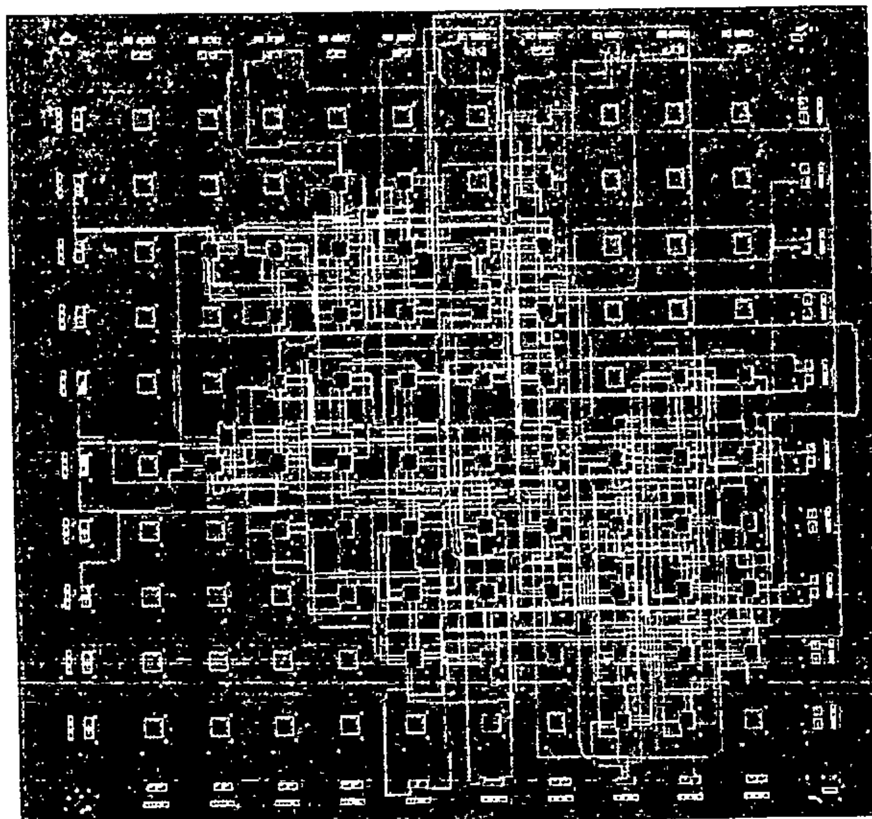


Figure 35. Circuit représentant le bloc SCALE

IV.1.7 La synthèse du bloc CORDIC

Le circuit cible utilisé est XC4006E-1-PQ160 FPGA.

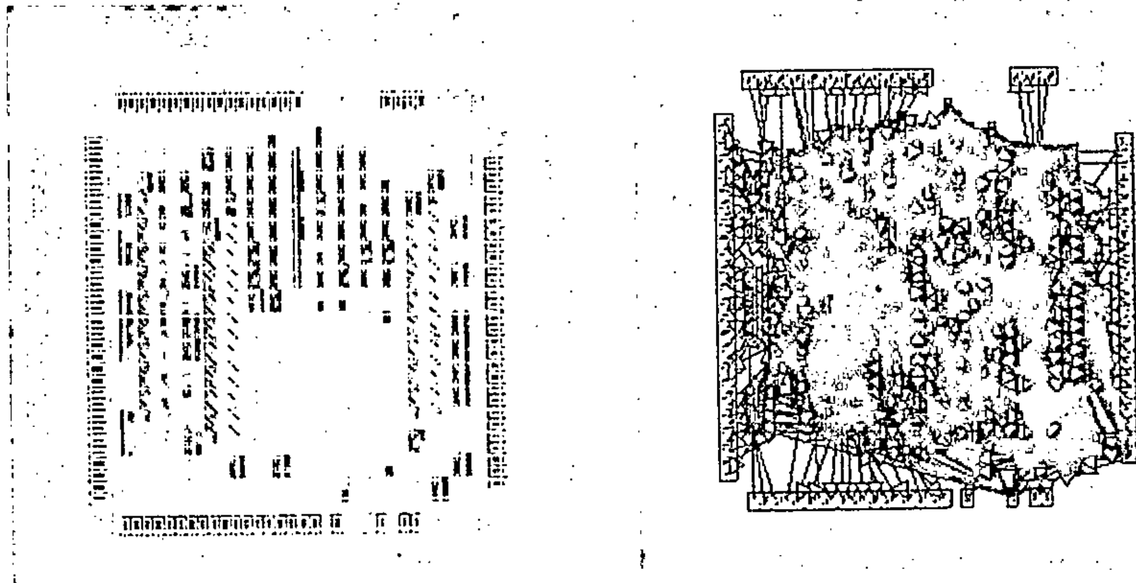


Figure 36. Le bloc CORDIC après la phase placement et routage

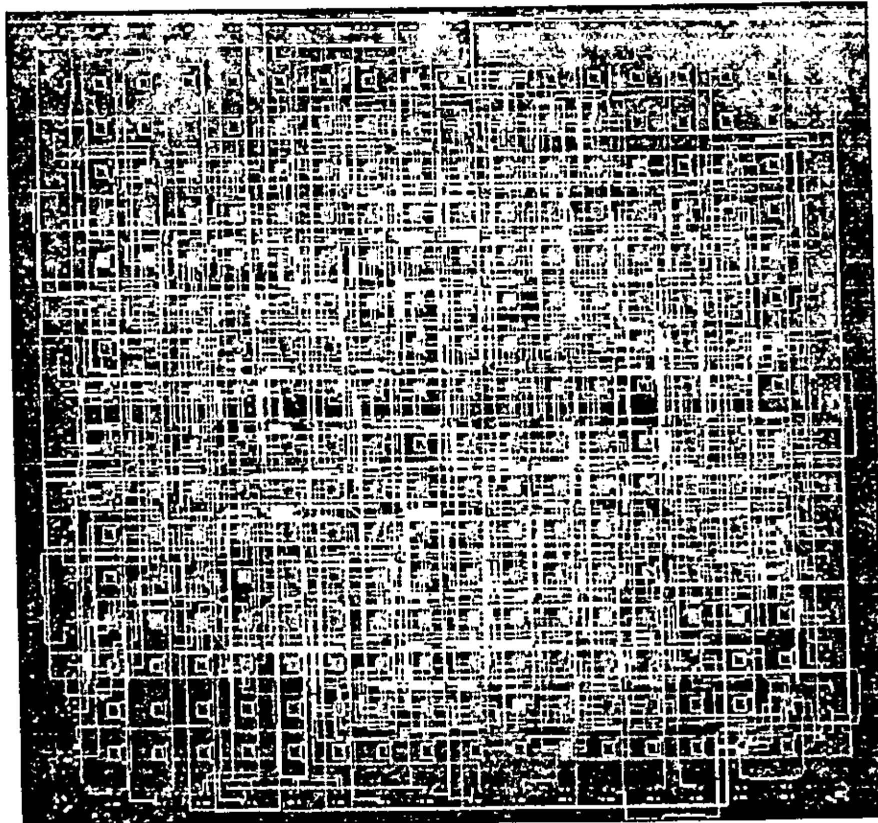


Figure 37. Circuit représentant le bloc CORDIC

	Nombre disponible	*Nombre utilisé*	*Pourcentage (%)*
CLBs	256	194	75%
Bascules des CLBs	512	53	10%
IBOs	128	102	76%
4 input LUTS	512	365	71%
3 input LUTS	256	24	9%
Nombre totale de portes logiques équivalentes : 3141			
Nombre de portes JTAG additionnelles pour les IOBs : 4896			
Chemins (Paths) : 19737			
Nœuds (Nets) : 447			
Connexions : 1404			
Période minimale : 47.921 ns			
Fréquence maximale : 20,868 MHz.			
Retard maximum engendré par un nœud :20.279 ns.			

Tableau 7. Les principaux résultats des rapports issus de la synthèse du bloc CORDIC

D'après la figure (37) et les résultats illustrés dans le tableau (7), nous pouvons dire que notre circuit est bien exploité.

Pour plus de détails concernant les différents rapports générés lors de la synthèse de chaque blocs, nous vous conseillons vivement de consulter l'*Annexe –E-*.

IV.2 Simulation temporelle

IV.2.1 Introduction

Tout circuit présente des retards ce qui ne l'empêche pas nécessairement de remplir sa fonction. Il nous faut donc préciser un peu a partir de quand la différence, entre le comportement idéal et la réalité, devient inacceptable. Une règle simple est de considérer comme inadmissible une différence qui subsiste lorsque survient un front d'horloge.

IV.2.2 Simulation temporelle du bloc de génération d'adresse :

Fréquence max : 33.422 MHz.

1)- **fréquence de simulation : 32 MHz** : remarquons que le plus grand retard par rapport au front montant est celui de la sortie FI, il est de 25,6ns , sachant qu'on est à la limite inférieur de la fréquence max.

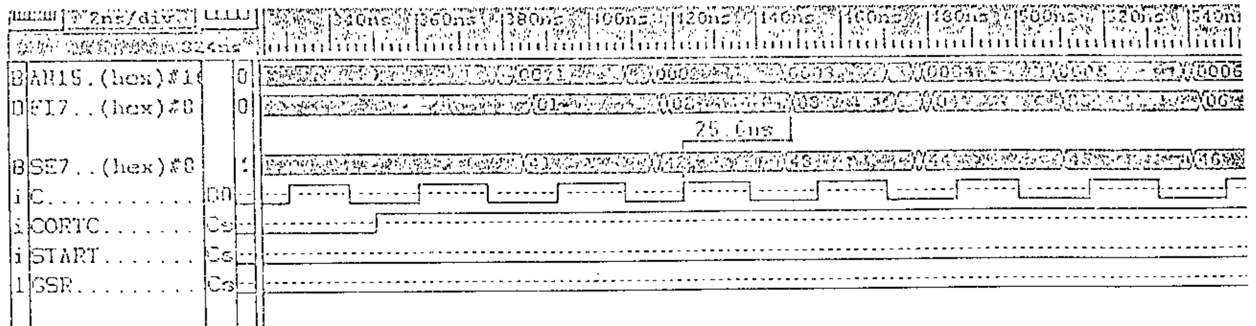


Figure 38. Chronogramme de la simulation temporelle du bloc de génération d'adresses à f=32MHz

2)- **fréquence de simulation : 35 MHz** : dépassant la fréquence max qui est de 33.422 MHz, les sorties sont presque toutes erronées et surtout celles qui présentent un retard supérieur à l'inverse de la fréquence max qui est de 29.92 ns.

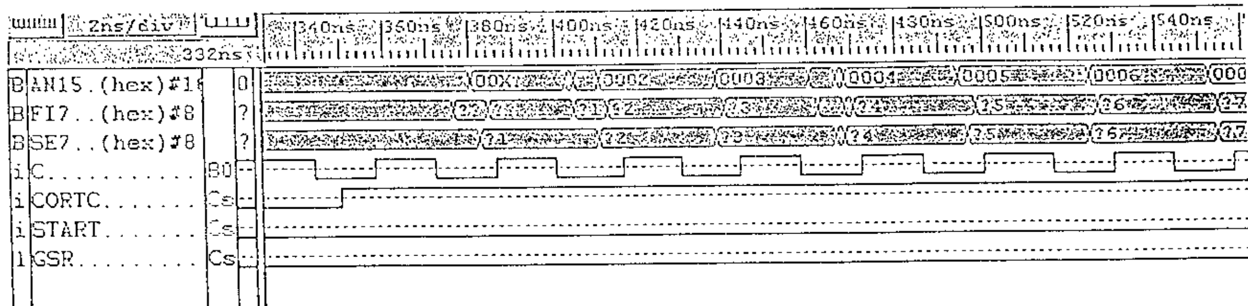


Figure 39. Chronogramme de la simulation temporelle du bloc de génération d'adresses à f=35MHz

IV.2.3 Simulation temporelle du bloc SCALE

Fréquence max :39,268 MHz

1)- fréquence de simulation : 38 MHz : remarquons que le plus grand retard par rapport au front montant est celui de la sortie OB, il est de 20ns , sachant qu'on est à la limite inférieur de la fréquence max.

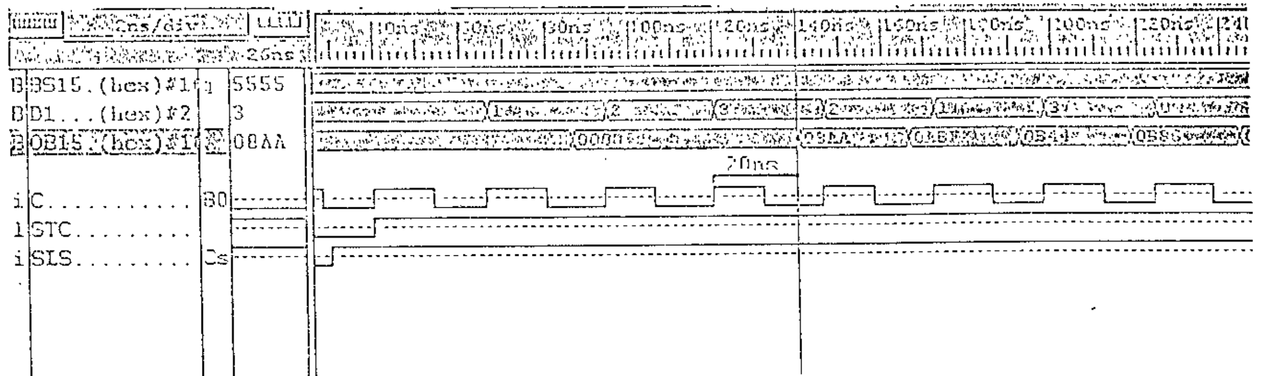


Figure 40. Chronogramme de la simulation temporelle du bloc SCALE à f=38MHz

2)- fréquence de simulation : 50 MHz : dépassant la fréquence max qui est de 39.268 MHz, les sorties sont presque toutes erronées et surtout celles qui présentent un retard supérieur à l'inverse de la fréquence max qui est de 25 ns.

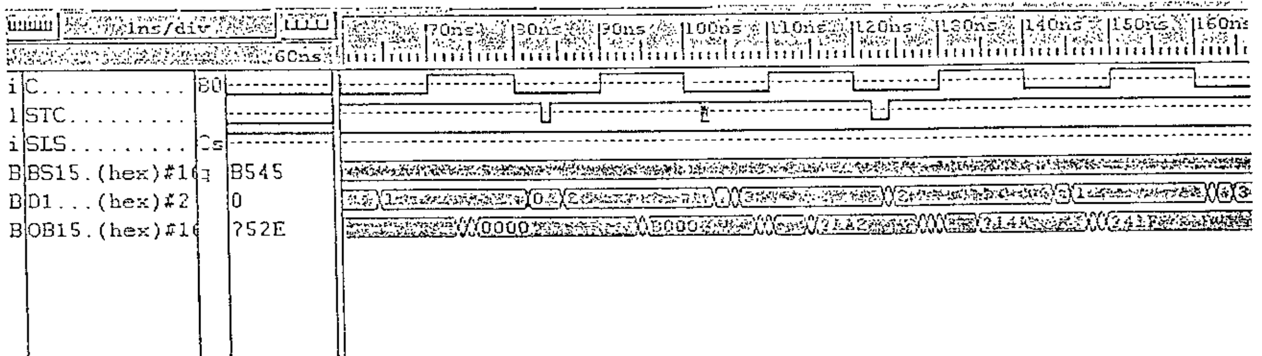


Figure 41 Chronogramme de la simulation temporelle du bloc SCALE à f=50MHz

IV.2.4 Simulation temporelle du bloc CORDIC

Fréquence max :20.868 MHz

1)- fréquence de simulation : 20 MHz : remarquons que le plus grand retard par rapport au front montant est celui de la sortie OY, il est de 38.1ns, sachant qu'on est à la limite inférieur de la fréquence max.

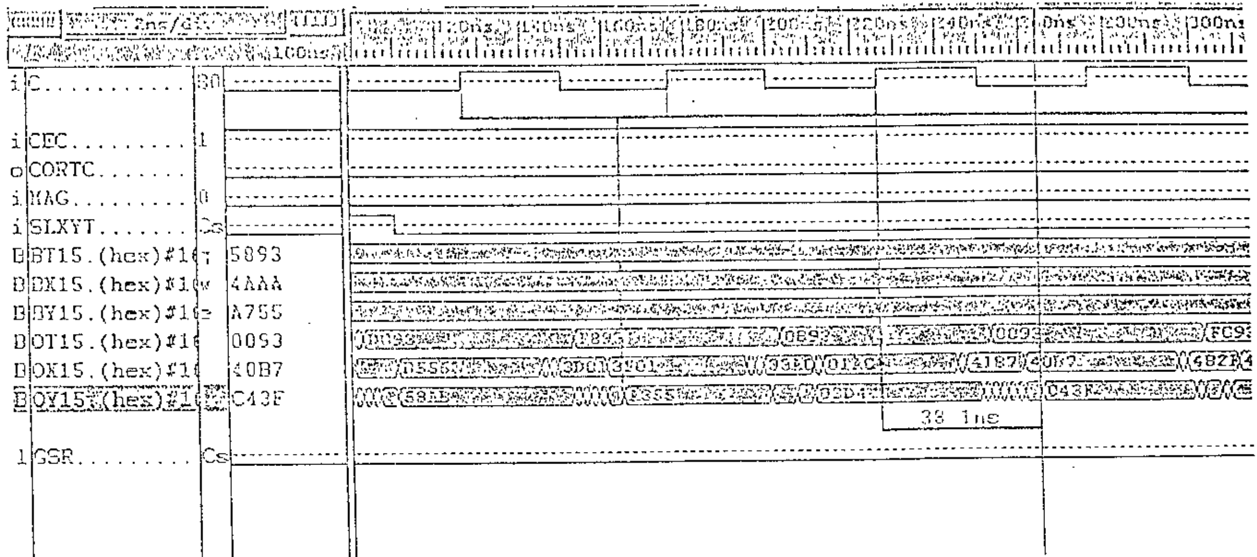


Figure 42 Chronogramme de la simulation temporelle du bloc CORDIC à f=20MHz

2)- fréquence de simulation : 25 MHz : dépassant la fréquence max qui est de 20.868 MHz, les sorties sont presque toutes erronées et surtout celles qui présentent un retard supérieur à l'inverse de la fréquence max qui est de 48 ns.

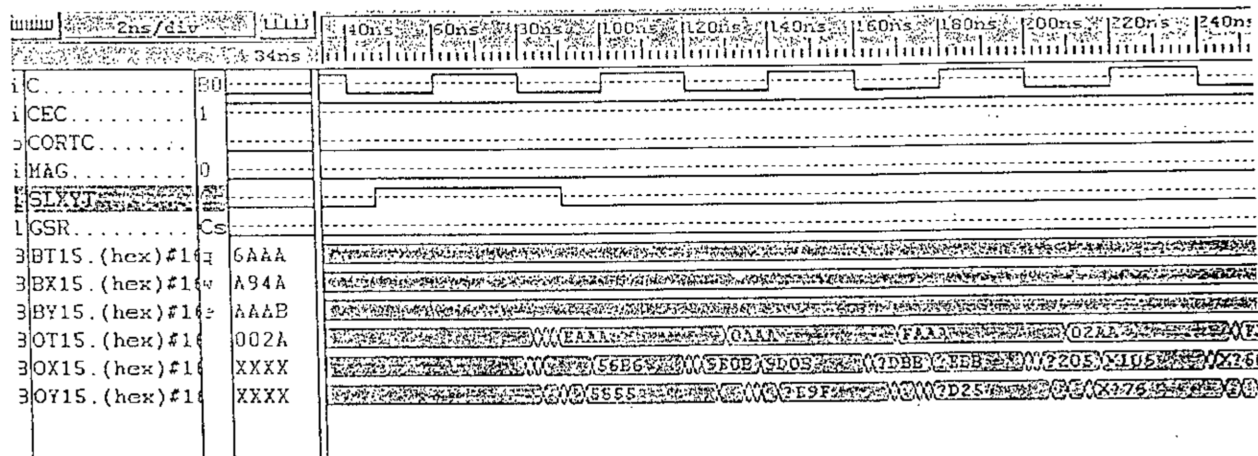


Figure 43 Chronogramme de la simulation temporelle du bloc CORDIC à f=25MHz

IV.3 Conclusion

Tout au long du travail, nous avons essayé d'identifier les structures et les syntaxes supportées correctement par le synthétiseur, mais également celles qui posent un problème.

Dans notre travail nous n'avons fait que simuler des algorithmes avec l'optimisation par défaut du synthétiseur du logiciel, mais nous pouvions imposer des contraintes temporelles et de répartition sur la surface, décrites par le cahier de charge, dans ce cas on doit faire ce qu'on appel une **rétroannotation** si le résultat n'est pas conforme aux espérances des performances du futur circuit. Quand le model correspondant n'est pas représentatif du programme ou du schéma source, dans le cas du VHDL il s'agit d'une description structurelle qui emploie des composants virtuels, représentés avec leurs retards. Ce modèle donne des résultats erronés en simulation quand les règles temporelles ne sont pas respectées.

V - Conclusion Générale

Depuis sa création, l'électronique intégrée s'efforce d'ajouter toujours de nouvelles fonctions dans un seul et même circuit. Selon la loi bien connue dictée par M.Moore dans les années soixante-dix, la complexité des circuits double tous les dix-huit mois.

Dans notre mémoire nous avons illustré une méthode de conception et nous avons détaillé sur un exemple concret les possibilités de codage offerte par l'outil de développement XILINX FOUNDATION ainsi que les performances proposées par les circuits FPGAs.

Nous avons également évoqué la notion de "design" multi-composants et de l'intérêt d'implémenter des fonctions et des programmes sur des circuits différents ce qui permet à titre d'exemple de décharger le microprocesseur de plusieurs fonctions souvent répétitives. Dans notre projet, nous avons conçu un processeur dédié au calcul de la FFT et qui peut être utilisé dans divers domaines comme le codage des signaux de parole, ou la détection des ondes sismiques, ou la surveillance des battements de cœur des nouveaux nés, ou au sein des appareilles de mesure de l'électronique de puissance ou encore pour le rehaussement d'un enregistrement musical par des effets sonores. Dans tous ces domaines, des analyseurs de spectres ont été choisis, basés sur l'emploi de la transformée de Fourier discrète (TFD) par les techniques numériques de transformation de Fourier rapide (FFT Fast Fourier Transforme) car l'algorithme de la FFT nous permet d'obtenir les résultats beaucoup plus rapidement; ce qui engendre un traitement en temps réel et la précision que nous offre cette méthode. Afin de calculer les termes de SINUS et de COSINUS de la FFT nous avons fait appel l'algorithme CORDIC.

La première phase de notre projet fut la rédaction des spécifications techniques et la conception de l'architecture selon la méthodologie "top-down"; ensuite, nous sommes passé à la simulation fonctionnelle des différents sous-blocs constituants notre architecture en utilisant la méthodologie "bottom-up".

Les différents modules, une fois, développés et validés, ont été synthétisés.

La synthèse et le placement-routage étant effectués, on disposait d'un plan du circuit tel qu'il sera fabriqué. On a pu alors extraire toutes les caractéristiques physiques du circuit réel comme la fréquence maximale de fonctionnement ainsi que les délais de propagation des signaux.

L'implémentation du schéma global n'a pu être faite du fait que le nombre de ressources sollicitées par notre application dépassent largement ceux du logiciel et c'est la raison pour laquelle nous avons choisi d'implémenter les trois blocs essentiels de la FFT séparément sur trois circuits cibles différents, simplement pour arriver à maîtriser tous les outils de travail inhérents aux étapes d'implémentation, puisque la philosophie de conception est la même.

Aussi nous pouvons utiliser ces blocs dans des applications différentes que ceux de la FFT, par exemple le bloc CORDIC peut etre utilisé dans la conception d'une calculatrice, d'un modulateur-démodulateur et bien d'autres applications.

Annexe A.

➤ Code VHD du bloc SUB3

```
--------------------------------------------------------------------------------
-- CONCEPTION : Le bloc SUB3.
-- BUT      : Conception d'un soustracteur de 3 bits.
-- Fichier  : Sub3.vhd
--------------------------------------------------------------------------------
--               Auteurs : BOUNOUA Amine & BOUSBIA Hind
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
------------------ Declaration de l'entity ----------------------------------------
entity sub3 is
  port (
       pt: in STD_LOGIC_vector(2 downto 0);
       ss: in STD_LOGIC_vector(2 downto 0);
       su: out STD_LOGIC_vector(2 downto 0)
       );
end sub3;

----------------- Architecture fonctionnelle ---------------------------------------
architecture sub3_arch of sub3 is
signal in1_signed,in2_signed,
  sub_signed : signed(2 downto 0);
begin
  in1_signed<=signed(pt);
  in2_signed<=signed(ss);
  process(in1_signed,in2_signed)
   begin
    sub_signed <= (in1_signed-in2_signed);
   end process ;
    su <= std_logic_vector(sub_signed);
end sub3_arch; -- Fin de l'architecture.
```

➤ Code VHD du bloc SHIFTER : bloc de Décalage.

```
--------------------------------------------------------------------------------
-- CONCEPTION : Le bloc SHIFTER.
-- BUT      : Conception d'un registre à décalage avec deux entrées.
--            Entrée des données sur 8 bits et une entrée sur 3 bits indiquant le nombre
--            de décalage.
-- Fichier  : Shifter1.vhd
--------------------------------------------------------------------------------
--               Auteurs : BOUNOUA Amine & BOUSBIA Hind
--------------------------------------------------------------------------------

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

```
-------------------- Déclaration de l'entité bshift2 --------------------------------
Entity bshift2 is
    port (
        shift  : in  std_logic_vector(2 downto 0);  -- Compteur de décalage.
        input  : in  std_logic_vector (7 downto 0);
        output  : out std_logic_vector (7 downto 0)
          );
end entity bshift2;

-------------------- Architecture fonctionnelle ------------------------------------
architecture behavior of bshift2 is
  function to_integer(sig : std_logic_vector) return integer is
   variable num : integer  ;
  begin
   for i in sig'range loop
    if sig(i)='1' then
     num := num*2+1;
    else
     num := num*2;
    end if;
   end loop;  -- fin de la boucle for.
   return num;
  end  to_integer;

begin
 amine:process(input, shift)
 begin
 case shift is
      when "000"=> output <=input;
      when "001"=>output <=  input(6 downto 0)&'0';
      when "010"=>output<=input(5 downto 0) & "00";
      when "011"=>output<= input(4 downto 0)&"000" ;
      when "100"=>output<= input(3 downto 0)&"0000" ;
      when "101"=>output<=input(2 downto 0) & "00000";
      when "110"=>output<= input(1 downto 0)& "000000";
      when "111"=>output<=  input(0 downto 0)&"0000000";

      when others=>output<="00000000";
         end case;
   end process amine;

end architecture behavior;
```

⏩ Code VHD du bloc NPT.

```
--------------------------------------------------------------------------------
-- CONCEPTION : Le bloc NPT.
-- BUT        : Sert a stocker le nombre 6.
-- Fichier    : Npt.vhd
--------------------------------------------------------------------------------
--          Auteurs :BOUNOUA Amine & BOUSBIA Hind
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity npt is
   port (
        so: out STD_LOGIC_vector(2 downto 0)
        );
end npt;
```

```
architecture npt_arch of npt is
begin
  so <= "110";
end npt_arch;
```

Code VHD du bloc INDEX.

```
---------------------------------------------------------------------------
-- Conception : INDEX (Entité & Architecture).
-- Fichier     : INDEX.vhd
-- BUT         : Registre Buffer pour le stockage de l'adresse du premier
--               opérande.
---------------------------------------------------------------------------
--              Auteurs : BOUSBIA Hind & BOUNOUA Amine.
---------------------------------------------------------------------------
Library IEEE;
Use IEEE.STD_LOGIC_1164.all;

Entity INDEX is
    Port ( Data_IN   : in std_logic_vector (7 downto 0);
           Clk       : in std_logic;
           CE        : in std_logic;
           Clear     : in std_logic;
           Data_Out  : out std_logic_vector (6 downto 0)
           );
End INDEX;
--------------------------- Architecture fonctionnelle -------------------

Architecture INDEX_ARCH of INDEX is

BEGIN
   P : process (Clk,Clear)
     Begin
       If Clear = '1' then
         Data_Out <= "0000000";
       elsif Clk'Event and Clk = '1' then
          if CE = '1' then
            for I in 0 to 6 loop
               Data_Out(I) <= Data_In(I);
            end loop;
          end if;
       end if;
    End process P;
END INDEX_ARCH;
```

Code VHD du bloc INCR3.

```
---------------------------------------------------------------------------------
-- CONCEPTION : Le bloc INCR3.
-- BUT        : Conception d'un compteur de trois bits.
-- Fichier    : Incr3.vhd
---------------------------------------------------------------------------------
--            Auteurs : BOUNOUA Amine & BOUSBIA Hind
---------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
-------------------- Déclaration de l'entité ---------------
entity incr3 is
   port (
        in1: in STD_LOGIC_vector(2 downto 0);
```

```
        ou: out STD_LOGIC_vector(2 downto 0)
        );
end incr3;

--------------------- Architecture fonctionnelle -------------
architecture incr3_arch of incr3 is
begin
   inc: process(in1)
       variable operand,increment:signed(2 downto 0);
       begin
        operand:=signed(in1);
        increment:=operand+1;
        ou <= std_logic_vector(increment);
       end process inc;
end incr3_arch;
```

⯮ Code VHD du bloc DECR8.

```
--------------------------------------------------------------------------------
-- CONCEPTION : Le bloc DECR8.
-- BUT        : Conception d'un dé compteur pour la décrémentation de 8 bits.
-- Fichier    : DECR8.vhd
--------------------------------------------------------------------------------
--               Auteurs :BOUNOUA Amine & BOUSBIA Hind
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
-------------- Declaration de l'entity ---------------
entity decr8 is
   port (
        in2: in STD_LOGIC_vector(7 downto 0);
        s  : out STD_LOGIC_vector(7 downto 0)
        );
end decr8;
------------- Architecture fonctionnelle -------------
architecture decr8_arch of decr8 is
begin
  dec:process(in2)
      variable operand,decrement:signed(7 downto 0);
       begin
        operand := signed(in2);
        decrement := operand-1;
        s <= std_logic_vector(decrement);
       end process dec;
end decr8_arch;
```

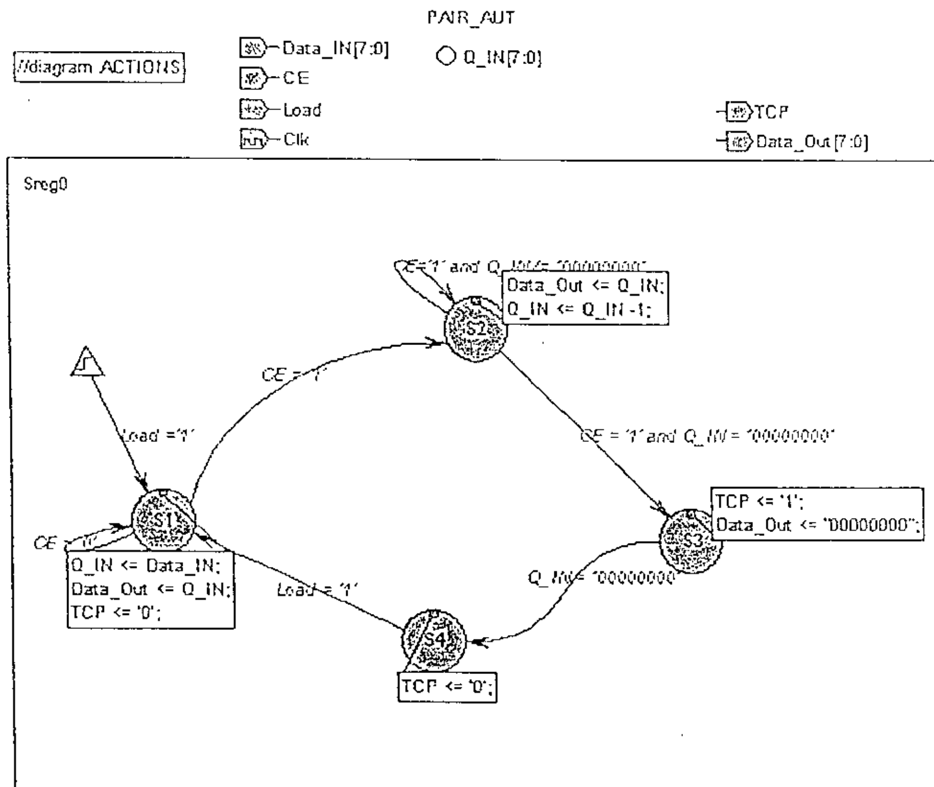
⯮ Code VHD du bloc PAIR

```
--
-- File: D:\ACTIVE\PROJECTS\GEN_ADDR\PAIR_AUT.vhd
-- created: 04/26/02 14:29:23
-- from: 'D:\ACTIVE\PROJECTS\GEN_ADDR\PAIR_AUT.asf'
-- by fsm2hdl - version: 2.0.1.49
--
library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity pair_aut is
  port (CE: in STD_LOGIC;
       Clk: in STD_LOGIC;
       Data_IN: in STD_LOGIC_VECTOR (7 downto 0);
       Load: in STD_LOGIC;
       Data_Out: out STD_LOGIC_VECTOR (7 downto 0);
       TCP: out STD_LOGIC);
end;

architecture pair_aut_arch of pair_aut is

--diagram signal declarations
signal Q_IN: STD_LOGIC_VECTOR (7 downto 0);

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (Clk)

begin

if Clk'event and Clk = '1' then
        if Load ='1' then
                Sreg0 <= S1;
                Q_IN <= Data_IN;
        else
        case Sreg0 is
                when S1 =>
                        if CE = '0' then
                                Sreg0 <= S1;
                                Q_IN <= Data_IN;
                        elsif CE = '1' then
                                Sreg0 <= S2;
                                Q_IN <= Q_IN -1;
                        end if;
                when S2 =>
                        if CE='1' and Q_IN /= "00000000" then
                                Sreg0 <= S2;
                                Q_IN <= Q_IN -1;
                        elsif CE = '1' and Q_IN = "00000000" then
                                Sreg0 <= S3;
                        end if;
                when S3 =>
                        if Q_IN = "00000000" then
                                Sreg0 <= S1;
                                Q_IN <= Data_IN;
                        end if;
                when others =>
                        null;
        end case;
        end if;
```

```
end if;
end process;

-- signal assignment statements for combinatorial outputs
Data_Out_assignment:
Data_Out <= Q_IN when (Sreg0 = S2) else
        "00000000" when (Sreg0 = S3) else
        Q_IN;

TCP_assignment:
TCP <= '1' when (Sreg0 = S3) else
     '0';

end pair_aut_arch;
```

→ FMS du bloc PAIR.



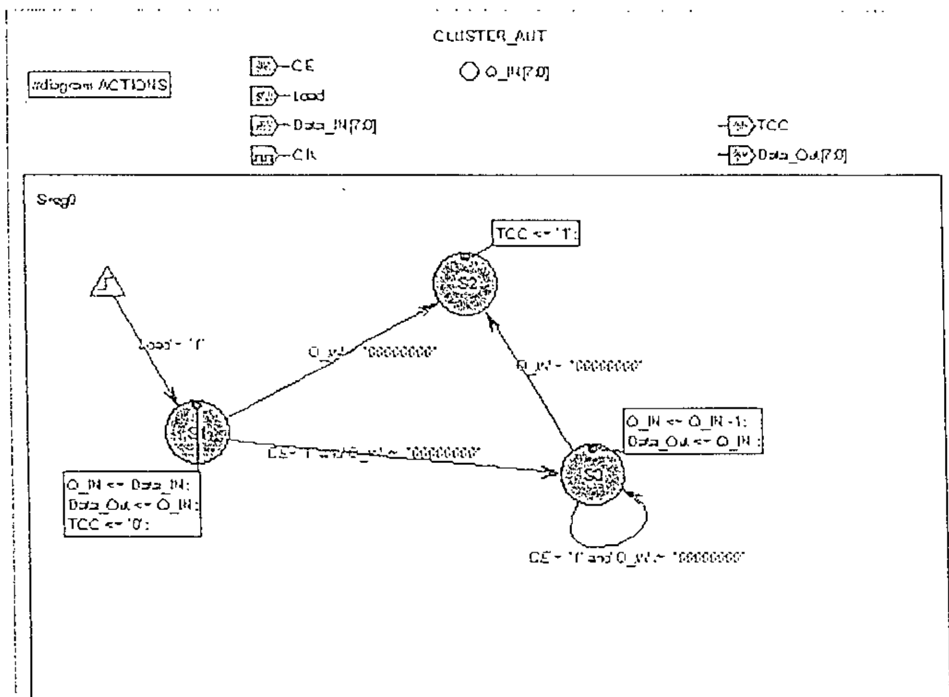
→ Code VHD du bloc CLUSTER.

```
--
-- File: D:\ACTIVE\PROJECTS\GEN_ADDR\CLUSTER_AUT.vhd
-- created: 04/26/02 14:39:43
-- from: 'D:\ACTIVE\PROJECTS\GEN_ADDR\CLUSTER_AUT.asf'
-- by fsm2hdl - version: 2.0.1.49
--
library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity cluster_aut is
  port (CE: in STD_LOGIC;
      Clk: in STD_LOGIC;
      Data_IN: in STD_LOGIC_VECTOR (7 downto 0);
      Load: in STD_LOGIC;
      Data_Out: out STD_LOGIC_VECTOR (7 downto 0);
      TCC: out STD_LOGIC);
end;

architecture cluster_aut_arch of cluster_aut is

--diagram signal declarations
signal Q_IN: STD_LOGIC_VECTOR (7 downto 0);

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (Clk)

begin

if Clk'event and Clk = '1' then
        if Load = '1' then
                Sreg0 <= S1;
                Q_IN <= Data_IN;
        else
        case Sreg0 is
                when S1 =>
                        if CE='1' and Q_IN /= "00000000" then
                                Sreg0 <= S3;
                                Q_IN <= Q_IN -1;
                        elsif Q_IN = "00000000" then
                                Sreg0 <= S2;
                        end if;
                when S2 =>
                when S3 =>
                        if CE = '1' and Q_IN /= "00000000" then
                                Sreg0 <= S3;
                                Q_IN <= Q_IN -1;
                        elsif Q_IN = "00000000" then
                                Sreg0 <= S2;
                        end if;
                when others =>
                        null;
        end case;
        end if;
 end if;
 end process;

 -- signal assignment statements for combinatorial outputs
 Data_Out_assignment:
```

```
Data_Out <= Q_IN when (Sreg0 = S3) else
        Q_IN;

TCC_assignment:
TCC <= '1' when (Sreg0 = S2) else
      '0';

end cluster_aut_arch;
```

➧ FSM du bloc CLUSTER.



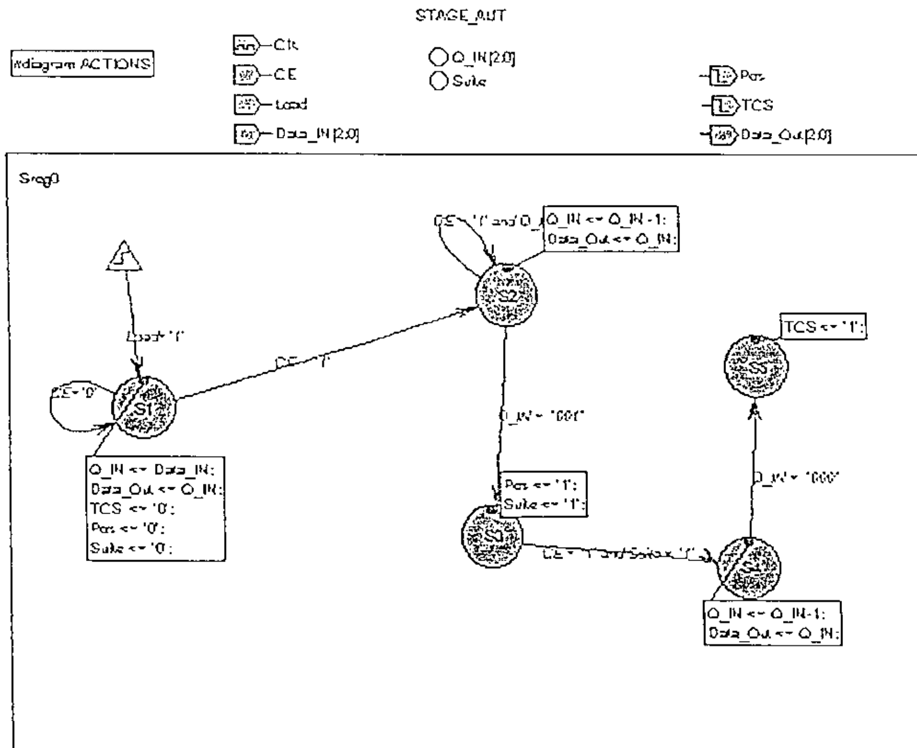
➧ Code VHD du bloc STAGE.

```
--
-- File: A:\STAGE_AUT.vhd
-- created: 04/27/02 04:44:37
-- from: 'A:\STAGE_AUT.asf'
-- by fsm2hdl - version: 2.0.1.49
--
library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity stage_aut is
  port (CE: in STD_LOGIC;
```

```
        Clk: in STD_LOGIC;
        Data_IN: in STD_LOGIC_VECTOR (2 downto 0);
        Load: in STD_LOGIC;
        Data_Out: out STD_LOGIC_VECTOR (2 downto 0);
        Pos: out STD_LOGIC;
        TCS: out STD_LOGIC);
end;

architecture stage_aut_arch of stage_aut is

--diagram signal declarations
signal Q_IN: STD_LOGIC_VECTOR (2 downto 0);
signal Suite: STD_LOGIC;

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3, S4, S5);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;

Sreg0_machine: process (Clk)
begin

if Clk'event and Clk = '1' then
        if Load='1' then
                Sreg0 <= S1;
                Q_IN <= Data_IN;
                TCS  <= '0';
                Pos <= '0';
                Suite <= '0';
        else
        case Sreg0 is
                when S1 =>
                        if CE = '1' then
                                Sreg0 <= S2;
                                Q_IN <= Q_IN -1;
                        elsif CE='0' then
                                Sreg0 <= S1;
                                Q_IN <= Data_IN;
                                TCS  <= '0';
                                Pos <= '0';
                                Suite <= '0';
                        end if;
                when S2 =>
                        if CE = '1' and Q_IN /="001" then
                                Sreg0 <= S2;
                                Q_IN <= Q_IN -1;
                        elsif Q_IN = "001" then
                                Sreg0 <= S3;
                                Pos <= '1';
                                Suite <= '1';
                        end if;
                when S3 =>
                        if CE = '1' and Suite = '1' then
                                Sreg0 <= S4;
                                Q_IN <= Q_IN-1;
                        end if;
                when S4 =>
```

```
                    if Q_IN = "000" then
                          Sreg0 <= S5;
                          TCS <= '1';
                    end if;
               when S5 =>
               when others =>
                    null;
          end case;
          end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
Data_Out_assignment:
Data_Out <= Q_IN when (Sreg0 = S2) else
        Q_IN when (Sreg0 = S4) else
        Q_IN;

end stage_aut_arch;
```

FSM du bloc STAGE.



Code VHD du bloc ANGLE.

```
--------------------------------------------------------------------
-- Conception : ANGLE (Entité & Architecture).
-- Fichier    : ANGLE.vhd
-- BUT        : Registre Buffer pour le stockage la valeur de l'angle k.
--------------------------------------------------------------------
--              Auteurs : BOUSBIA Hind & BOUNOUA Amine
--------------------------------------------------------------------

Library IEEE;
Use IEEE.STD_LOGIC_1164.all;
```

```
Entity ANGLE is
    Port ( Data_IN  : in std_logic_vector (7 downto 0);
           Clk      : in std_logic;
           CE       : in std_logic;
           Clear    : in std_logic;
           Data_Out : out std_logic_vector (15 downto 0)
          );
End ANGLE;
-------------------------- Architecture fonctionnelle ------------------

Architecture ANGLE_ARCH of ANGLE is

BEGIN
   P : process (Clk, Clear)
    Begin
      If Clear = '1' then
        Data_Out <= (others =>'0');
      elsif Clk'Event and Clk = '1' then
          if CE = '1' then
            Data_Out <= (others =>'0');
            for I in 0 to 7 loop
               Data_Out(I) <= Data_In(I);
            end loop;
          end if;
      end if;
   End process P;
END ANGLE_ARCH;
```

Code VHD du bloc INDEX.

```
---------------------------------------------------------------------------
-- Conception : INDEX (Entité & Architecture).
-- Fichier     : INDEX.vhd
-- BUT         : Registre Buffer pour le stockage de l'adresse du premier
--               opérande.
---------------------------------------------------------------------------
--               Auteurs : BOUSBIA Hind & BOUNOUA Amine.
---------------------------------------------------------------------------
Library IEEE;
Use IEEE.STD_LOGIC_1164.all;

Entity INDEX is
    Port ( Data_IN  : in std_logic_vector (7 downto 0);
           Clk      : in std_logic;
           CE       : in std_logic;
           Clear    : in std_logic;
           Data_Out : out std_logic_vector (6 downto 0)
          );
End INDEX;
-------------------------- Architecture fonctionnelle ------------------
Architecture INDEX_ARCH of INDEX is

BEGIN
   P : process (Clk,Clear)
    Begin
      If Clear = '1' then
        Data_Out <= "0000000";
      elsif Clk'Event and Clk = '1' then
          if CE = '1' then
            for I in 0 to 6 loop
```

```
            Data_Out(I) <= Data_In(I);
          end loop;
        end if;
      end if;
    End process P;
END INDEX_ARCH;
```

Annexe – B –

Le bloc SHIFTER1

```
--------------------------------------------------------------------------------
-- CONCEPTION : Registre décaleur d'un bit.
-- BUT        : faire décaler une valeur de 16 bits d'un bit à droite.
-- Fichier    : Shifter.vhd
--------------------------------------------------------------------------------
--          Auteurs : BOUNOUA Amine & BOUSBIA Hind.
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
------------- Déclaration de l'entity ---------------
Entity bshift2 is
    port (
        d0      : in std_logic ;       -- activation de decalage.
        input   : in  std_logic_vector (15 downto 0);
        output  : out std_logic_vector (15 downto 0) );
end entity bshift2;

------------- Architecture fonctionnelle -------------
architecture behavior of bshift2 is


begin
  amine: process(input, d0)
  begin
  case d0 is
      when '0'=> output <=input;
      when '1'=>output <= ('0' & input(15 downto 1));

       when others => output<="0000000000000000";
         end case;
         end process amine;

  end architecture behavior;
```

Le bloc SHIFTER2

```
--------------------------------------------------------------------------------
-- CONCEPTION : Registre décaleur de deux  bits.
-- BUT        : faire décaler une valeur de 16 bits de deux  bits à droite.
-- Fichier    : Shifter2.vhd
--------------------------------------------------------------------------------
--          Auteurs : BOUNOUA Amine & BOUSBIA Hind.
--------------------------------------------------------------------------------
```

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

Entity bshift2 is
    port (
        d1       : in std_logic ;
        input    : in  std_logic_vector (15 downto 0);
        output   : out std_logic_vector (15 downto 0) );
end entity bshift2;


architecture behavior of bshift2 is


begin  -- behavior
  amine : process(input, d1)
  begin
  case d1 is
      when '0'=> output <=input;
      when '1'=>output <= ("00" & input(15 downto 2));

       when others=>output<="0000000000000000";
       end case;
       end process amine;

 end architecture behavior;  -- of bshift
```

Le bloc MUXREG16

```
-------------------------------------------------------------------------------------
-- CONCEPTION : Registre Multiplexeur (16 bits) de deux entrées vers une sortie.
--.
--
-- Fichier    : MUX.vhd
-------------------------------------------------------------------------------------
--          Auteurs : BOUNOUA Amine & BOUSBIA Hind.
-------------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
------------- Déclaration de l'entity ---------------
entity mux_reg is
   port (
      a: in STD_LOGIC_vector(15 downto 0);
      b: in STD_LOGIC_vector(15 downto 0);
      ce:in std_logic;
      sl: in STD_LOGIC;
      clk: in STD_LOGIC;
```

```
    reset: in STD_LOGIC;
    s: out STD_LOGIC_vector(15 downto 0)
  );
end mux_reg;
-------------------- Architecture fonctionnelle -----------------
architecture mux_reg_arch of mux_reg is
component mux is
port(i1,i2,sel: in std_logic;out_mux_1: out std_logic);
end component;
signal out_mux,in_reg:std_logic_vector(15 downto 0);
begin

gen : for i in 0 to 15 generate
instance: mux port map (i1=>a(i),i2=>b(i),sel=>sl,out_mux_1=>out_mux(i));
end generate gen;
sequencial : process(reset,clk, in_reg,ce)
begin
if (clk'event and clk='1') then
if ce ='1' then
if reset='1' then
in_reg<=(others=>'0');
else
in_reg <= out_mux;
end if;
end if;
end if;
s <= in_reg;
end process sequencial;
end mux_reg_arch;
```

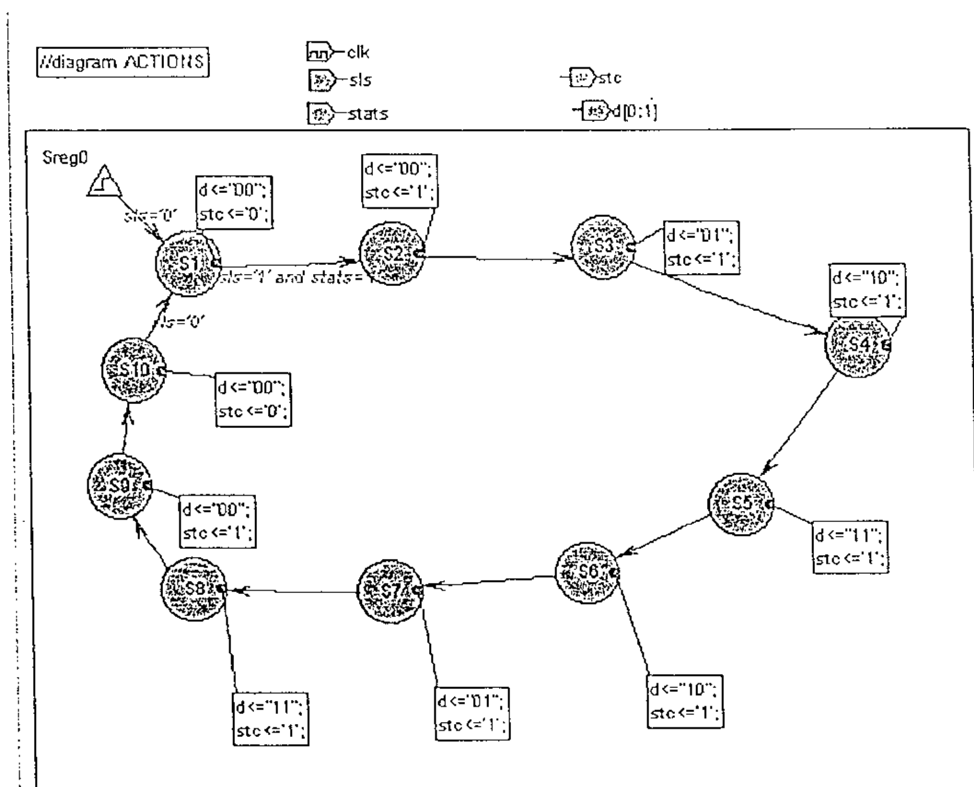
Le bloc STATUS

```
--------------------------------------------------------------------------------
-- CONCEPTION : STATUS.
--
-- Fichier    : STATUS.vhd
--------------------------------------------------------------------------------
--          Auteurs : BOUSBIA Hind & BOUNOUA Amine.
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity status1 is
   port (
      sls  : in STD_LOGIC;
      stc  : in STD_LOGIC;
      clk : in STD_LOGIC;
      stats: out STD_LOGIC
   );
end status1;
```

```
architecture status1_arch of status1 is
begin
combinatoire : process (clk, sls, stc)
begin
 if (sls='0') then
 stats <= '0';
 elsif (clk'event and clk='1') then
  if (stc='1') then
    stats <= '0';
  else
    stats <= '1';
  end if;
 end if;
end process combinatoire;
end status1_arch;
```

Le bloc CO

(a) - Le graphe FSM.



(b) - Le code VHDL

```
--
-- File: C:\FNDTN\ACTIVE\PROJECTS\SCALE\co.vhd
-- created: 05/31/02 07:49:50
-- from: 'C:\FNDTN\ACTIVE\PROJECTS\SCALE\co.ASF'
-- by fsm2hdl - version: 2.0.1.49
--
library IEEE;
```

```
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity co is
 port (clk: in STD_LOGIC;
      sls: in STD_LOGIC;
      stats: in STD_LOGIC;
      d: out STD_LOGIC_VECTOR (0 to 1);
      stc: out STD_LOGIC);
end;

architecture co_arch of co is


-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S10, S2, S3, S4, S5, S6, S7, S8, S9);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (clk)

begin

if clk'event and clk = '1' then
        if sls='0' then
                Sreg0 <= S1;
        else
        case Sreg0 is
                when S1 =>
                        if sls='1' and stats='1' then
                                Sreg0 <= S2;
                        end if;
                when S10 =>
                        if sls='0' then
                                Sreg0 <= S1;
                        end if;
                when S2 =>
                        Sreg0 <= S3;
                when S3 =>
                        Sreg0 <= S4;
                when S4 =>
```

```
                Sreg0 <= S5;
            when S5 =>
                Sreg0 <= S6;
            when S6 =>
                Sreg0 <= S7;
            when S7 =>
                Sreg0 <= S8;
            when S8 =>
                Sreg0 <= S9;
            when S9 =>
                Sreg0 <= S10;
            when others =>
                null;
        end case;
        end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
d_assignment:
    d <= "00" when (Sreg0 = S10) else
   "00" when (Sreg0 = S2) else
   "01" when (Sreg0 = S3) else
   "10" when (Sreg0 = S4) else
   "11" when (Sreg0 = S5) else
   "10" when (Sreg0 = S6) else
   "01" when (Sreg0 = S7) else
   "11" when (Sreg0 = S8) else
   "00" when (Sreg0 = S9) else
   "00";

stc_assignment:
    stc <= '0' when (Sreg0 = S10) else
    '1' when (Sreg0 = S2) else
    '1' when (Sreg0 = S3) else
    '1' when (Sreg0 = S4) else
    '1' when (Sreg0 = S5) else
    '1' when (Sreg0 = S6) else
    '1' when (Sreg0 = S7) else
    '1' when (Sreg0 = S8) else
    '1' when (Sreg0 = S9) else
    '0';

end co_arch;
```

Annexe -C-

➤ Code VHDL du bloc muxN

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity muxN is
   port (
      e1: in STD_LOGIC_vector(15 downto 0);
      e2: in STD_LOGIC_vector(15 downto 0);
      ce1:in std_logic;
      sele: in STD_LOGIC;
      clk:in std_logic;
      reset: in STD_LOGIC;
      r:out std_logic;
      out_n: out STD_LOGIC_vector(15 downto 0)
       );
end muxN;

architecture muxN_arch of muxN is
component mux_reg
port(a,b:in std_logic_vector(15 downto 0);
ce:in std_logic;
sl:in std_logic;
clk:in std_logic; reset: in std_logic;
s:out std_logic_vector(15 downto 0));
end component;
signal e :std_logic_vector(15 downto 0);
begin
mux_t:mux_reg port map(a=>e,b=>e2,ce=>ce1,sl=>sele,clk=>clk
,reset=>reset,s=>out_n);
muxreg: process(e1)
 variable n1: std_logic_vector(15 downto 0);
 variable n2: std_logic_vector(15 downto 0);

 begin
 n1:=e1;
 r<=e1(15);

 n2:=
n1(14)&n1(14)&n1(13)&n1(12)&n1(11)&n1(10)&n1(9)&n1(8)&n1(7)&n1(6)&n1(5)&
n1(4)&n1(3)&n1(2)&n1(1)&n1(0);

 e <= std_logic_vector(n2);
```

```
end process muxreg;
end  muxN_arch ;
```

Code VHDL du bloc THETA

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity theta is
   port (
          e: in STD_LOGIC_vector(3 downto 0);
          s: out STD_LOGIC_vector(15 downto 0)
        );
end theta;

architecture theta_arch of theta is
begin
process (e)
begin
case e is
        when "0000" =>s<= "0010000000000000";
        when "0001" =>s<= "0001000000000000";
        when "0010" =>s<="0000100000000000";
        when "0011"=>s<= "0000010000000000";
        when "0100"=>s<="0000001000000000";
        when "0101"=>s<="0000000100000000";
        when "0110"=>s<="0000000010000000";
        when "0111"=>s<="0000000001000000";
        when "1000"=>s<="0000000000100000";
        when "1001"=>s<="0000000000010000";
        when "1010"=>s<="0000000000001000";
        when "1011"=>s<="0000000000000100";
        when "1100"=>s<="0000000000000010";
        when "1101"=>s<="0000000000000001";

        when others=>s<="0000000000000000";
 end case;
 end process;
 end theta_arch;
```

Code VHDL du bloc mux_reg

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux_reg is
   port (
      a: in STD_LOGIC_vector(15 downto 0);
      b: in STD_LOGIC_vector(15 downto 0);
```

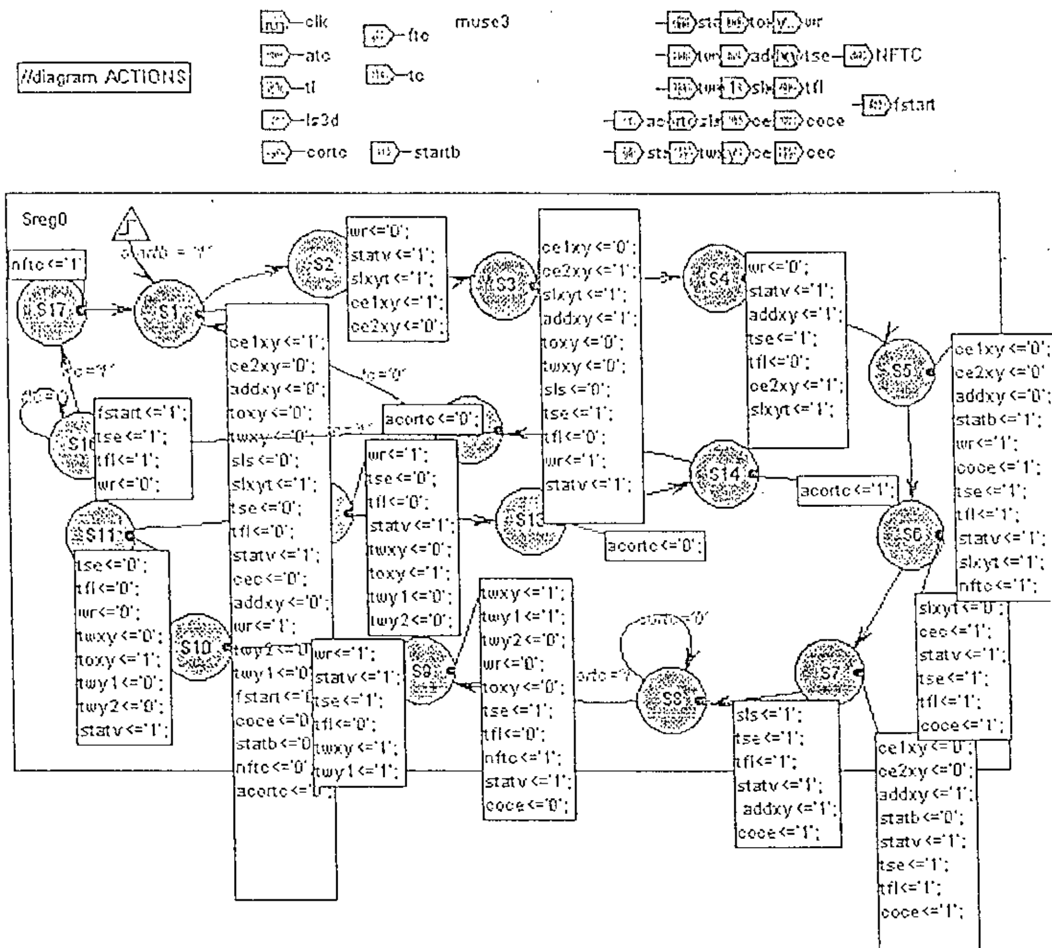
```
      ce:in std_logic;
      sl    : in STD_LOGIC;
      clk   : in STD_LOGIC;
      reset: in STD_LOGIC;
      s: out STD_LOGIC_vector(15 downto 0)
   );
end mux_reg;

architecture mux_reg_arch of mux_reg is
component mux is
port(i1,i2,sel:in std_logic;out_mux_1:out std_logic);
end component;
signal out_mux,in_reg:std_logic_vector(15 downto 0);
begin

gen: for i in 0 to 15 generate
instance: mux port map (i1=>a(i),i2=>b(i),sel=>sl,out_mux_1=>out_mux(i));
end generate gen;
sequencial:process(reset,clk,in_reg,ce)
begin
if (clk'event and clk='1') then
        if ce='1' then
           if reset='1' then
                in_reg <= (others => '0');

           else
           in_reg<=out_mux;
          end if;
      end if;
end if;
      s <= in_reg;
end process sequencial;
end mux_reg_arch;
```

▸▸ Code VHDL du bloc Masse

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
entity zero is
   port (
            boub : out STD_LOGIC_VECTOR (15 downto 0)
          );
end zero;

architecture zero_ARCH of zero is

begin
       boub <= "0000000000000000";
end zero_ARCH;
```

Annexe -D-

La machine d'états finis(FSM) du bloc contrôleur.



Le code VHDL du bloc Contrôleur.

```
--
-- File: D:\ACTIVE\PROJECTS\CONTROLL\muse3.vhd
-- created: 05/04/02 13:13:57
-- from: 'D:\ACTIVE\PROJECTS\CONTROLL\muse3.ASF'
-- by fsm2hdl - version: 2.0.1.49
--
library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;
```

```
entity muse3 is
 port (atc: in STD_LOGIC;
      clk: in STD_LOGIC;
      cortc: in STD_LOGIC;
      ftc: in STD_LOGIC;
      ls3d: in STD_LOGIC;
      startb: in STD_LOGIC;
      tc: in STD_LOGIC;
      tl: in STD_LOGIC;
      acortc: out STD_LOGIC;
      addxy: out STD_LOGIC;
      ce1xy: out STD_LOGIC;
      ce2xy: out STD_LOGIC;
      cec: out STD_LOGIC;
      coce: out STD_LOGIC;
      fstart: out STD_LOGIC;
      NFTC: out STD_LOGIC;
      sls: out STD_LOGIC;
      slxyt: out STD_LOGIC;
      statb: out STD_LOGIC;
      statv: out STD_LOGIC;
      tfl: out STD_LOGIC;
      toxy: out STD_LOGIC;
      tse: out STD_LOGIC;
      twxy: out STD_LOGIC;
      twy1: out STD_LOGIC;
      twy2: out STD_LOGIC;
      wr: out STD_LOGIC);
end;

architecture muse3_arch of muse3 is


-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S10, S11, S12, S13, S14, S15, S16, S17, S2, S3, S4, S5, S6, S7, S8,
S9);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (clk)

begin

if clk'event and clk = '1' then
        if startb = '1' then
                Sreg0 <= S1;
```

```
else
case Sreg0 is
        when S1 =>



                Sreg0 <= S2;
        when S10 =>
                Sreg0 <= S11;
        when S11 =>
                Sreg0 <= S12;
        when S12 =>
                Sreg0 <= S13;
        when S13 =>
                Sreg0 <= S14;
        when S14 =>
                Sreg0 <= S15;
        when S15 =>
                if tc='1' then
                        Sreg0 <= S16;
                elsif tc='0' then
                        Sreg0 <= S1;
                end if;
        when S16 =>
                if ftc='1' then
                        Sreg0 <= S17;
                elsif ftc='0' then
                        Sreg0 <= S16;
                end if;
        when S17 =>
                Sreg0 <= S1;
        when S2 =>
                Sreg0 <= S3;
        when S3 =>
                Sreg0 <= S4;
        when S4 =>
                Sreg0 <= S5;
        when S5 =>
                Sreg0 <= S6;
        when S6 =>
                Sreg0 <= S7;
        when S7 =>
                Sreg0 <= S8;
        when S8 =>
                if cortc='0' then
                        Sreg0 <= S8;
                elsif cortc='1' then
                        Sreg0 <= S9;
```

```
                end if;
            when S9 =>
                Sreg0 <= S10;
            when others =>
                null;
        end case;
        end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
ce1xy_assignment:
ce1xy <= '1' when (Sreg0 = S2) else
      '0' when (Sreg0 = S3) else
      '0' when (Sreg0 = S5) else
      '0' when (Sreg0 = S7) else
      '1';
ce2xy_assignment:
ce2xy <= '0' when (Sreg0 = S2) else
      '1' when (Sreg0 = S3) else
      '1' when (Sreg0 = S4) else
      '0' when (Sreg0 = S5) else
      '0' when (Sreg0 = S7) else
      '0';
addxy_assignment:
addxy <= '1' when (Sreg0 = S3) else
      '1' when (Sreg0 = S4) else
      '0' when (Sreg0 = S5) else
      '1' when (Sreg0 = S7) else
      '1' when (Sreg0 = S8) else
      '0';
toxy_assignment:
toxy <= '1' when (Sreg0 = S11) else
      '1' when (Sreg0 = S12) else
      '0' when (Sreg0 = S3) else
      '0' when (Sreg0 = S9) else
      '0';
twxy_assignment:
twxy <= '1' when (Sreg0 = S10) else
      '0' when (Sreg0 = S11) else
      '0' when (Sreg0 = S12) else
      '0' when (Sreg0 = S3) else
      '1' when (Sreg0 = S9) else
      '0';

sls_assignment:
sls <= '0' when (Sreg0 = S3) else
      '1' when (Sreg0 = S8) else
      '0';
slxyt_assignment:
```

```
slxyt <= '1' when (Sreg0 = S2) else
        '1' when (Sreg0 = S3) else
        '1' when (Sreg0 = S4) else
        '1' when (Sreg0 = S5) else
        '0' when (Sreg0 = S6) else
        '1';
tsc_assignment:
tsc <= '1' when (Sreg0 = S10) else
      '0' when (Sreg0 = S11) else
      '0' when (Sreg0 = S12) else
      '1' when (Sreg0 = S16) else
      '1' when (Sreg0 = S3) else
      '1' when (Sreg0 = S4) else
      '1' when (Sreg0 = S5) else
      '1' when (Sreg0 = S6) else
      '1' when (Sreg0 = S7) else
      '1' when (Sreg0 = S8) else
      '1' when (Sreg0 = S9) else
      '0';
tfl_assignment:
tfl <= '0' when (Sreg0 = S10) else
      '0' when (Sreg0 = S11) else
      '0' when (Sreg0 = S12) else
      '1' when (Sreg0 = S16) else
      '0' when (Sreg0 = S3) else
      '0' when (Sreg0 = S4) else
      '1' when (Sreg0 = S5) else
      '1' when (Sreg0 = S6) else
      '1' when (Sreg0 = S7) else
      '1' when (Sreg0 = S8) else
      '0' when (Sreg0 = S9) else
      '0';
statv_assignment:
statv <= '1' when (Sreg0 = S10) else
        '1' when (Sreg0 = S11) else
        '1' when (Sreg0 = S12) else
        '1' when (Sreg0 = S2) else
        '1' when (Sreg0 = S3) else
        '1' when (Sreg0 = S4) else
        '1' when (Sreg0 = S5) else
        '1' when (Sreg0 = S6) else
        '1' when (Sreg0 = S7) else
        '1' when (Sreg0 = S8) else
        '1' when (Sreg0 = S9) else
        '1';
ccc_assignment:
ccc <= '1' when (Sreg0 = S6) else
      '0';
wr_assignment:
wr <= '1' when (Sreg0 = S10) else
```

```
'0' when (Sreg0 = S11) else
'1' when (Sreg0 = S12) else
'0' when (Sreg0 = S16) else
'0' when (Sreg0 = S2) else
'1' when (Sreg0 = S3) else
'0' when (Sreg0 = S4) else
'1' when (Sreg0 = S5) else
'0' when (Sreg0 = S9) else
'1';
twy2_assignment:
twy2 <= '0' when (Sreg0 = S11) else
      '0' when (Sreg0 = S12) else
      '0' when (Sreg0 = S9) else
      '0';

twy1_assignment:
twy1 <= '1' when (Sreg0 = S10) else
      '0' when (Sreg0 = S11) else
      '0' when (Sreg0 = S12) else
      '1' when (Sreg0 = S9) else
      '0';
fstart_assignment:
fstart <= '1' when (Sreg0 = S16) else
      '0';
coce_assignment:
coce <= '1' when (Sreg0 = S5) else
      '1' when (Sreg0 = S6) else
      '1' when (Sreg0 = S7) else
      '1' when (Sreg0 = S8) else
      '0' when (Sreg0 = S9) else
      '0';
statb_assignment:
statb <= '1' when (Sreg0 = S5) else
      '0' when (Sreg0 = S7) else
      '0';
nftc_assignment:
nftc <= '1' when (Sreg0 = S17) else
      '1' when (Sreg0 = S5) else
      '1' when (Sreg0 = S9) else
      '0';

acortc_assignment:
acortc <= '0' when (Sreg0 = S13) else
      '1' when (Sreg0 = S14) else
      '0' when (Sreg0 = S15) else
      '0';

end muse3_arch;
```

Annexe — E —

I - Les rapports issus de la synthèse du bloc de génération d'adresse

I.1- Le rapport de conversion

```
ngdbuild:  version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Command Line: ngdbuild -p xc4003e-1-pc84 -uc
c:\fndtn\active\projects\redompt\redompt.ucf -dd ..
c:\fndtn\active\projects\redompt\redompt.edn redompt.ngd

NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   0

Writing NGD file "redompt.ngd" ...

Writing NGDBUILD log file "redompt.bld"...
```

Le rapport de mapping:

```
              Xilinx Mapping Report File for Design "redompt"
          Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line   : map -p xc4003e-1-pc84 -o map.ncd redompt.ngd redompt.pcf
Target Device  : x4003e
Target Package : pc84
Target Speed   : -1
Mapper Version : xc4000e -- M1.5.19
Mapped Date    : Wed Jun 05 05:06:51 2002

Design Summary
--------------
   Number of errors:         0
   Number of warnings:      14
   Number of CLBs:              97 out of   100   97%
      CLB Flip Flops:       80
      4 input LUTs:        178 (3 used as route-throughs)
      3 input LUTs:         33 (7 used as route-throughs)
   Number of bonded IOBs:       36 out of    61   59%
      IOB Flops:             0
      IOB Latches:           0
   Number of RPM macros:         7
   11 unrelated functions packed into 7 CLBs.
   (7% of the CLBs used are affected.)
```

```
Total equivalent gate count for design: 1990
Additional JTAG gate count for IOBs:     1728
```

1.2- Le rapport de placement-routage

```
PAR: Xilinx Place And Route M1.5.19.
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Wed Jun 05 05:06:56 2002

par -w -ol 2 -d 0 map.ncd redompt.ncd redompt.pcf


Constraints file: redompt.pcf

Loading device database for application par from file "map.ncd".
   "redompt" is an NCD, version 2.27, device xc4003e, package pc84, speed -
1
Loading device for application par from file '4003e.nph' in environment
C:/fndtn.
Device speed data version:  x1_0.95  PRELIMINARY.

Device utilization summary:

   Number of External IOBs               36 out of 61      59%
      Flops:                              0
      Latches:                            0

   Number of CLBs                        97 out of 100     97%
      Total CLB Flops:                   80 out of 200     40%
      4 input LUTs:                     178 out of 200     89%
      3 input LUTs:                      33 out of 100     33%

 Finished Constructive Placer.  REAL time: 2 secs

 Writing design to file "redompt.ncd".

 Starting Optimizing Placer.  REAL time: 2 secs
 Optimizing
 Swapped 6 comps.
 Xilinx Placer [1]   49980   REAL time: 2 secs

 Finished Optimizing Placer.  REAL time: 2 secs

 Writing design to file "redompt.ncd".

 Total REAL time to Placer completion: 2 secs
 Total CPU time to Placer completion: 0 secs
 Total REAL time to Router completion: 5 secs
 Total CPU time to Router completion: 0 secs

 Generating PAR statistics.

    The Delay Summary Report

    The Score for this design is: 471


 The Number of signals not completely routed for this design is: 0
```

```
The Average Connection Delay for this design is:        2.951 ns
The Average Connection Delay on critical nets is:       0.000 ns
The Average Clock Skew for this design is:             17.976 ns
The Maximum Pin Delay is:                              20.194 ns
The Average Connection Delay on the 10 Worst Nets is:   8.802 ns

Listing Pin Delays by value: (ns)

 d <= 10    < d <= 20   < d <= 30   < d <= 40   < d <= 50    d > 50
---------   ---------   ---------   ---------   ---------   ---------
      662          24           1           0           0           0

Writing design to file "redompt.ncd".

All signals are completely routed.

Total REAL time to PAR completion: 5 secs
Total CPU time to PAR completion: 0 secs

PAR done.
```

I.3- Le rapport des pins

```
PAR: Xilinx Place And Route M1.5.19.
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
Wed Jun 05 05:07:01 2002

Xilinx PAD Specification File
*****************************

Input file:       map.ncd
Output file:      redompt.ncd
Part type:        xc4003e
Speed grade:      -1
Package:          pc84

Wed Jun 05 05:07:01 2002

Pinout by Pin Name:
```

Pin Name	Direction	Pin Number
AN<0>	OUTPUT	P83
AN<10>	OUTPUT	P36
AN<11>	OUTPUT	P29
AN<12>	OUTPUT	P38
AN<13>	OUTPUT	P81
AN<14>	OUTPUT	P9
AN<15>	OUTPUT	P70

```
| AN<1>                                              | OUTPUT   | P23
|
| AN<2>                                              | OUTPUT   | P72
|
| AN<3>                                              | OUTPUT   | P78
|
| AN<4>                                              | OUTPUT   | P79
|
| AN<5>                                              | OUTPUT   | P80
|
| AN<6>                                              | OUTPUT   | P71
|
| AN<7>                                              | OUTPUT   | P69
|
| AN<8>                                              | OUTPUT   | P15
|
| AN<9>                                              | OUTPUT   | P14
|
| C                                                  | INPUT    | P8
|
| CORTC                                              | INPUT    | P40
|
| FI<0>                                              | OUTPUT   | P48
|
| FI<1>                                              | OUTPUT   | P61
|
| FI<2>                                              | OUTPUT   | P68
|
| FI<3>                                              | OUTPUT   | P3
|
| FI<4>                                              | OUTPUT   | P41
|
| FI<5>                                              | OUTPUT   | P59
|
| FI<6>                                              | OUTPUT   | P45
|
| FI<7>                                              | OUTPUT   | P44
|
| SE<0>                                              | OUTPUT   | P46
|
| SE<1>                                              | OUTPUT   | P47
|
| SE<2>                                              | OUTPUT   | P60
|
| SE<3>                                              | OUTPUT   | P65
|
| SE<4>                                              | OUTPUT   | P66
|
| SE<5>                                              | OUTPUT   | P62
|
| SE<6>                                              | OUTPUT   | P67
|
| SE<7>                                              | OUTPUT   | P82
|
| START                                              | INPUT    | P7
|
| TC                                                 | OUTPUT   | P84
|
+----------------------------------------------------+----------+------------
LWed Jun 05 05:07:01 2002
```

File: redompt.dly

1.4- Le rapport de retard asynchrone:

The 20 Worst Net Delays are:

Max Delay (ns)	Netname
20.194	$Net00154_
9.571	SH<2>
9.505	$Net00141_
7.783	SS<0>
7.521	SH<1>
7.412	CO<1>
6.953	SH<0>
6.650	IN<4>
6.630	&__A__18
6.408	$Net00133_
6.350	CO<5>
6.225	NP<2>
5.808	U12/syn864
5.785	CO<3>
5.741	$Net00149_
5.737	SS<2>
5.330	$Net00143_
4.998	CO<4>
4.960	SS<1>
4.871	&__A__17

1.5- Le rapport temporelle Post Layout

Xilinx TRACE, Version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc. All rights reserved.

Design file: redompt.ncd
Physical constraint file: redompt.pcf
Device,speed: xc4003e,-1 (x1_0.95 PRELIMINARY)
Report level: error report, limited to 3 items per constraint
Timing summary:

Timing errors: 0 Score: 0

Constraints cover 3035 paths, 234 nets, and 685 connections (100.0% coverage)

Design statistics:
Minimum period: 29.920ns (Maximum frequency: 33.422MHz)
Maximum net delay: 20.194ns

WARNING:bastw:544 - Clock nets using non-dedicated resources were found in this design. Clock skew on these resources will not be automatically addressed during path analysis. To create a timing report that analyzes clock skew for these paths, run trce with the '-skew' option.

Analysis completed Wed Jun 05 05:07:03 2002

I.6- Le rapport de génération de bits

```
Loading device database for application Bitgen from file "redompt.ncd".
   "redompt" is an NCD, version 2.27, device xc4003e, package pc84, speed -
1
Loading device for application Bitgen from file '4003e.nph' in environment
C:/fndtn.
Opened constraints file redompt.pcf.

BITGEN: Xilinx Bitstream Generator M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Wed Jun 05 05:07:16 2002

bitgen -l -w -g ConfigRate:SLOW -g TdoPin:PULLNONE -g M1Pin:PULLNONE -g
DonePin:PULLUP -g CRC:enable -g StartUpClk:CCLK -g SyncToDone:no -g
DoneActive:C1 -g OutputsActive:C3 -g GSRInactive:C4 -g ReadClk:CCLK -g
ReadCapture:enable -g ReadAbort:disable -g M0Pin:PULLNONE -g M2Pin:PULLNONE
redompt.ncd

Running DRC.
DRC detected 0 errors and 0 warnings.
Saving ll file in "redompt.ll".
Creating bit map...
Saving bit stream in "redompt.bit".
```

II - *Les rapport issus de la synthèse du bloc CORDIC*

II.1- Le rapport de conversion

```
ngdbuild:  version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Command Line: ngdbuild -p xc4006e-1-pq160 -uc
c:\fndtn\active\projects\shecore\shecore.ucf -dd ..
c:\fndtn\active\projects\shecore\shecore.edn shecore.ngd
Checking expanded design ...
WARNING:basnu:113 - logical net "$I6/CEO" has no load

NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   1

Writing NGD file "shecore.ngd" ...

Writing NGDBUILD log file "shecore.bld"...
```

II.2- Le rapport de mapping

```
  Xilinx Mapping Report File for Design "shecore"
         Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line   : map -p xc4006e-1-pq160 -o map.ncd shecore.ngd shecore.pcf
Target Device  : x4006e
```

```
Target Package : pq160
Target Speed   : -1
Mapper Version : xc4000e -- M1.5.19
Mapped Date    : Wed Jun 05 02:39:55 2002

Design Summary
--------------
   Number of errors:          0
   Number of warnings:        8
   Number of CLBs:              194 out of   256   75%
      CLB Flip Flops:        53
      4 input LUTs:         365
      3 input LUTs:          24
   Number of bonded IOBs:       102 out of   128   79%
      IOB Flops:              0
      IOB Latches:            0
   Number of RPM macros:          5
Total equivalent gate count for design: 3141
Additional JTAG gate count for IOBs:    4896
```

II.3- Le rapport de placement-routage

```
PAR: Xilinx Place And Route M1.5.19.
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Wed Jun 05 02:39:58 2002

par -w -ol 2 -d 0 map.ncd shecore.ncd shecore.pcf

Constraints file: shecore.pcf

Loading device database for application par from file "map.ncd".
   "shecore" is an NCD, version 2.27, device xc4006e, package pq160, speed
-1
Loading device for application par from file '4006e.nph' in environment
C:/fndtn.
Device speed data version:  x1_0.95  PRELIMINARY.

Device utilization summary:

   Number of External IOBs               102 out of 128    79%
      Flops:                               0
      Latches:                             0

   Number of CLBs                        194 out of 256    75%
      Total CLB Flops:                    53 out of 512    10%
      4 input LUTs:                      365 out of 512    71%
      3 input LUTs:                       24 out of 256     9%

Writing design to file "shecore.ncd".

Starting Optimizing Placer.  REAL time: 8 secs
Optimizing
Swapped 26 comps.
Xilinx Placer [1]   111668   REAL time: 9 secs

Finished Optimizing Placer.  REAL time: 9 secs

Writing design to file "shecore.ncd".
```

```
Total REAL time to Placer completion: 9 secs
Total CPU time to Placer completion: 0 secs

Total REAL time to Router completion: 20 secs
Total CPU time to Router completion: 0 secs

Generating PAR statistics.

   The Delay Summary Report

   The Score for this design is: 743

The Number of signals not completely routed for this design is: 0

   The Average Connection Delay for this design is:         4.611 ns
   The Average Connection Delay on critical nets is:        0.000 ns
   The Average Clock Skew for this design is:              17.620 ns
   The Maximum Pin Delay is:                               20.279 ns
   The Average Connection Delay on the 10 Worst Nets is:   14.128 ns

   Listing Pin_Delays by value: (ns)

    d <= 10    < d <= 20   < d <= 30   < d <= 40   < d <= 50    d > 50
   ---------   ---------   ---------   ---------   ---------   ---------
        1334         100           1           0           0           0

Writing design to file "shecore.ncd".

All signals are completely routed.

Total REAL time to PAR completion: 20 secs
Total CPU time to PAR completion: 0 secs

PAR done.
```

II.4- Le rapport des pins

```
PAR: Xilinx Place And Route M1.5.19.
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
Wed Jun 05 02:40:18 2002

Xilinx PAD Specification File
*****************************

Input file:        map.ncd
Output file:       shecore.ncd
Part type:         xc4006e
Speed grade:       -1
Package:           pq160

Wed Jun 05 02:40:18 2002

Pinout by Pin Name:
+-------------------------------------------------+-----------+-------------
-+
|                    Pin Name                     | Direction |  Pin Number
|
+-------------------------------------------------+-----------+-------------
-+
| BT<0>                                           |  INPUT    | P34
|
```

Signal	Type	Pin
BT<10>	INPUT	P12
BT<11>	INPUT	P13
BT<12>	INPUT	P158
BT<13>	INPUT	P159
BT<14>	INPUT	P5
BT<15>	INPUT	P154
BT<1>	INPUT	P44
BT<2>	INPUT	P33
BT<3>	INPUT	P45
BT<4>	INPUT	P43
BT<5>	INPUT	P25
BT<6>	INPUT	P24
BT<7>	INPUT	P17
BT<8>	INPUT	P16
BT<9>	INPUT	P15
BX<0>	INPUT	P87
BX<10>	INPUT	P103
BX<11>	INPUT	P112
BX<12>	INPUT	P113
BX<13>	INPUT	P114
BX<14>	INPUT	P111
BX<15>	INPUT	P115
BX<1>	INPUT	P89
BX<2>	INPUT	P88
BX<3>	INPUT	P86
BX<4>	INPUT	P93
BX<5>	INPUT	P96
BX<6>	INPUT	P94
BX<7>	INPUT	P106
BX<8>	INPUT	P128

BX<9>	INPUT	P127
BY<0>	INPUT	P49
BY<10>	INPUT	P146
BY<11>	INPUT	P145
BY<12>	INPUT	P8
BY<13>	INPUT	P153
BY<14>	INPUT	P152
BY<15>	INPUT	P149
BY<1>	INPUT	P36
BY<2>	INPUT	P56
BY<3>	INPUT	P57
BY<4>	INPUT	P62
BY<5>	INPUT	P59
BY<6>	INPUT	P46
BY<7>	INPUT	P31
BY<8>	INPUT	P26
BY<9>	INPUT	P3
C	INPUT	P69
CEC	INPUT	P85
CORTC	OUTPUT	P65
MAG	INPUT	P155
OT<0>	OUTPUT	P32
OT<10>	OUTPUT	P11
OT<11>	OUTPUT	P7
OT<12>	OUTPUT	P6
OT<13>	OUTPUT	P156
OT<14>	OUTPUT	P4
OT<15>	OUTPUT	P157
OT<1>	OUTPUT	P30
OT<2>	OUTPUT	P35

Signal	Direction	Pin
OT<3>	OUTPUT	P37
OT<4>	OUTPUT	P28
OT<5>	OUTPUT	P23
OT<6>	OUTPUT	P22
OT<7>	OUTPUT	P21
OT<8>	OUTPUT	P18
OT<9>	OUTPUT	P14
OVF	OUTPUT	P9
OZ<0>	OUTPUT	P73
OX<10>	OUTPUT	P108
OX<11>	OUTPUT	P105
OX<12>	OUTPUT	P130
OX<13>	OUTPUT	P107
OX<14>	OUTPUT	P129
OX<15>	OUTPUT	P109
OX<1>	OUTPUT	P90
OX<2>	OUTPUT	P72
OX<3>	OUTPUT	P92
OX<4>	OUTPUT	P95
OX<5>	OUTPUT	P98
OX<6>	OUTPUT	P97
OX<7>	OUTPUT	P102
OX<8>	OUTPUT	P99
OX<9>	OUTPUT	P104
OY<0>	OUTPUT	P48
OY<10>	OUTPUT	P150
OY<11>	OUTPUT	P148
OY<12>	OUTPUT	P143
OY<13>	OUTPUT	P147
OY<14>	OUTPUT	P144

```
| OY<15>                                          |  OUTPUT    | P140
|
| OY<1>                                           |  OUTPUT    | P63
|
| OY<2>                                           |  OUTPUT    | P50
|
| OY<3>                                           |  OUTPUT    | P53
|
| OY<4>                                           |  OUTPUT    | P47
|
| OY<5>                                           |  OUTPUT    | P58
|
| OY<6>                                           |  OUTPUT    | P52
|
| OY<7>                                           |  OUTPUT    | P54
|
| OY<8>                                           |  OUTPUT    | P27
|
| OY<9>                                           |  OUTPUT    | P55
|
| SLXYT                                           |  INPUT     | P84
|
+-------------------------------------------------+------------+-------------
```

II.5- Le rapport de retard asynchrone

```
Wed Jun 05 02:40:18 2002

File: shecore.dly

 The 20 Worst Net Delays are:
-------------------------------
| Max Delay (ns)  | Netname    |
-------------------------------
  20.279            $Net00277_
  18.769            COU<2>
  17.948            $Net00281_
  13.568            COU<3>
  13.010            $Net00269_
  13.007            COU<0>
  12.152            $Net00275_
  11.210            COU<1>
  10.928            QY<15>
  10.411            U8/C91
  10.086            U8/C95
   9.811            QX<14>
   9.623            $Net00282_
   9.531            QX<7>
   9.488            QY<3>
   9.223            QY<13>
   9.114            U8/C75
   8.856            QX<11>
   8.793            U9/C95
   8.742            QY<11>
```

II.6- Le rapport temporel Post Layout

```
--------------------------------------------------------------------------------
Xilinx TRACE, Version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
```

```
Design file:                shecore.ncd
Physical constraint file: shecore.pcf
Device,speed:               xc4006e,-1 (x1_0.95  PRELIMINARY)
Report level:               error report, limited to 3 items per constraint
--------------------------------------------------------------------------------

================================================================================
Timing constraint: Default period analysis
 19737 items analyzed, 0 timing errors detected.
 Minimum period is  47.921ns.
 Maximum delay is  56.157ns.
--------------------------------------------------------------------------------

================================================================================
Timing constraint: Default net enumeration
 447 items analyzed, 0 timing errors detected.
 Maximum net delay is  20.279ns.
--------------------------------------------------------------------------------
Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 19737 paths, 447 nets, and 1404 connections (100.0%
coverage)

Design statistics:
   Minimum period:  47.921ns (Maximum frequency:  20.868MHz)
   Maximum combinational path delay:  56.157ns
   Maximum net delay:  20.279ns

Analysis completed Wed Jun 05 02:40:21 2002
```

```
Le rapport de génération de bits :
Loading device database for application Bitgen from file "shecore.ncd".
   "shecore" is an NCD, version 2.27, device xc4006e, package pq160, speed
-1
Loading device for application Bitgen from file '4006e.nph' in environment
C:/fndtn.
Opened constraints file shecore.pcf.


BITGEN: Xilinx Bitstream Generator M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Wed Jun 05 02:40:23 2002

bitgen -l -w -g ConfigRate:SLOW -g TdoPin:PULLNONE -g M1Pin:PULLNONE -g
DonePin:PULLUP -g CRC:enable -g StartUpClk:CCLK -g SyncToDone:no -g
DoneActive:C1 -g OutputsActive:C3 -g GSRInactive:C4 -g ReadClk:CCLK -g
ReadCapture:enable -g ReadAbort:disable -g M0Pin:PULLNONE -g M2Pin:PULLNONE
shecore.ncd
Running DRC.
DRC detected 0 errors and 0 warnings.
Saving ll file in "shecore.ll".
Creating bit map...
Saving bit stream in "shecore.bit".
```

III - Les rapports issus de la synthèse du bloc SCALE

III.1- Le rapport de conversion

```
ngdbuild:  version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Command Line: ngdbuild -p xc4003e-1-pc84 -uc
c:\fndtn\active\projects\scale\scale.ucf -dd ..
c:\fndtn\active\projects\scale\scale.edn scale.ngd
```

III.2- Le rapport de mapping

```
          Xilinx Mapping Report File for Design "scale"
          Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line    : map -p xc4003e-1-pc84 -o map.ncd scale.ngd scale.pcf
Target Device   : x4003e
Target Package  : pc84
Target Speed    : -1
Mapper Version  : xc4000e -- M1.5.19
Mapped Date     : Wed Jun 05 03:00:33 2002

Design Summary
--------------
   Number of errors:         0
   Number of warnings:       3
   Number of CLBs:              54 out of   100   54%
      CLB Flip Flops:       46
      4 input LUTs:         99
      3 input LUTs:          7 (1 used as route-throughs)
   Number of bonded IOBs:       34 out of    61   55%
      IOB Flops:             0
      IOB Latches:           0
   Number of RPM macros:         1
Total equivalent gate count for design: 1002
Additional JTAG gate count for IOBs:    1632
Le rapport de placement-routage :

Copyright (c) 1995-1998
PAR: Xilinx Place And Route M1.5.19.
 Xilinx, Inc.  All rights reserved.

Wed Jun 05 03:00:38 2002

par -w -ol 2 -d 0 map.ncd scale.ncd scale.pcf
Device utilization summary:

   Number of External IOBs             34 out of 61      55%
      Flops:                            0
      Latches:                          0

   Number of CLBs                      54 out of 100     54%
      Total CLB Flops:                 46 out of 200     23%
      4 input LUTs:                    99 out of 200     49%
      3 input LUTs:                     7 out of 100      7%
```

```
Starting initial Placement phase.  REAL time: 0 secs
Finished initial Placement phase.  REAL time: 0 secs
Writing design to file "scale.ncd".

Starting Optimizing Placer.  REAL time: 0 secs
Optimizing
Swapped 6 comps.
Xilinx Placer [1]   23220   REAL time: 0 secs

Finished Optimizing Placer.  REAL time: 0 secs

Writing design to file "scale.ncd".

Total REAL time to Placer completion: 2 secs
Total CPU time to Placer completion: 0 secs

Total REAL time to Router completion: 3 secs
Total CPU time to Router completion: 0 secs

Generating PAR statistics.
   The Delay Summary Report
   The Score for this design is: 374

The Number of signals not completely routed for this design is: 0

   The Average Connection Delay for this design is:         2.554 ns
   The Average Connection Delay on critical nets is:        0.000 ns
   The Average Clock Skew for this design is:              11.897 ns
   The Maximum Pin Delay is:                               14.152 ns
   The Average Connection Delay on the 10 Worst Nets is:    5.940 ns

   Listing Pin Delays by value: (ns)

    d <= 10    < d <= 20   < d <= 30   < d <= 40   < d <= 50    d > 50
   ---------   ---------   ---------   ---------   ---------   ---------
         350           7           0           0           0           0

Writing design to file "scale.ncd".
All signals are completely routed.
Total REAL time to PAR completion: 3 secs
Total CPU time to PAR completion: 0 secs
PAR done.
```

III.3- Le rapport de pins

```
PAR: Xilinx Place And Route M1.5.19.
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
Wed Jun 05 03:00:41 2002

Xilinx PAD Specification File
*****************************
Input file:        map.ncd
Output file:       scale.ncd
Part type:         xc4003e
Speed grade:       -1
Package:           pc84

Wed Jun 05 03:00:41 2002
```

Pinout by Pin Name:

Pin Name	Direction	Pin Number
BS<0>	INPUT	P40
BS<10>	INPUT	P7
BS<11>	INPUT	P6
BS<12>	INPUT	P5
BS<13>	INPUT	P4
BS<14>	INPUT	P17
BS<15>	INPUT	P16
BS<1>	INPUT	P41
BS<2>	INPUT	P44
BS<3>	INPUT	P45
BS<4>	INPUT	P84
BS<5>	INPUT	P3
BS<6>	INPUT	P24
BS<7>	INPUT	P23
BS<8>	INPUT	P25
BS<9>	INPUT	P20
C	INPUT	P49
OB<0>	OUTPUT	P59
OB<10>	OUTPUT	P80
OB<11>	OUTPUT	P69
OB<12>	OUTPUT	P82
OB<13>	OUTPUT	P81
OB<14>	OUTPUT	P83
OB<15>	OUTPUT	P78
OB<1>	OUTPUT	P60
OB<2>	OUTPUT	P47
OB<3>	OUTPUT	P46

OB<4>	OUTPUT	P61
OB<5>	OUTPUT	P62
OB<6>	OUTPUT	P65
OB<7>	OUTPUT	P68
OB<8>	OUTPUT	P66
OB<9>	OUTPUT	P67
SLS	INPUT	P79

III.4- Le rapport de retard asynchrone

Wed Jun 05 03:00:41 2002

File: scale.dly

The 20 Worst Net Delays are:

Max Delay (ns)	Netname
14.152	$Net00111_
9.568	$Net00137_
6.933	$Net00001_
6.042	D<0>
4.786	STC
4.387	D<1>
3.950	A<9>
3.263	B<7>
3.232	A<6>
3.176	A<7>
3.162	U8/Sreg0<4>
3.087	U8/n_681
3.054	$Net00110_
2.976	B<9>
2.902	B<11>
2.900	A<4>
2.828	A<14>
2.718	B<5>
2.690	U8/Sreg0<7>
2.686	A<8>

III.5- Le rapport temporel Post Layout

```
--------------------------------
Xilinx TRACE, Version M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
Design file:                scale.ncd
Physical constraint file:  scale.pcf
Device,speed:               xc4003e,-1 (x1_0.95  PRELIMINARY)
Report level:               error report, limited to 3 items per constraint
--------------------------------------------------------------------------
==========================================================================
Timing constraint: Default period analysis
 2870 items analyzed, 0 timing errors detected.
 Minimum period is  25.466ns.
```

```
--------------------------------------------------------------------------------

================================================================================
Timing constraint: Default net enumeration
 128 items analyzed, 0 timing errors detected.
 Maximum net delay is  14.152ns.
--------------------------------------------------------------------------------
Timing summary:
---------------
Timing errors: 0  Score: 0

Constraints cover 2870 paths, 128 nets, and 356 connections (100.0%
coverage)

Design statistics:
   Minimum period:  25.466ns (Maximum frequency:  39.268MHz)
   Maximum net delay:  14.152ns

Analysis completed Wed Jun 05 03:00:45 2002
```

III.6- Le rapport de génération de bits

```
Loading device database for application Bitgen from file "scale.ncd".
   "scale" is an NCD, version 2.27, device xc4003e, package pc84, speed -1
Loading device for application Bitgen from file '4003e.nph' in environment
C:/fndtn.
Opened constraints file scale.pcf.


BITGEN: Xilinx Bitstream Generator M1.5.19
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Wed Jun 05 03:00:49 2002

bitgen -l -w -g ConfigRate:SLOW -g TdoPin:PULLNONE -g M1Pin:PULLNONE -g
DonePin:PULLUP -g CRC:enable -g StartUpClk:CCLK -g SyncToDone:no -g
DoneActive:C1 -g OutputsActive:C3 -g GSRInactive:C4 -g ReadClk:CCLK -g
ReadCapture:enable -g ReadAbort:disable -g M0Pin:PULLNONE -g M2Pin:PULLNONE
scale.ncd

Running DRC.
DRC detected 0 errors and 0 warnings.
Saving ll file in "scale.ll".
Creating bit map...
Saving bit stream in "scale.bit".
```

Glossaire.

ASIC *(Application Specific Integrated Circuit) : Circuit intégré sur-mesure pour les besoins d'un client ou d'une application.*

CAO : *Conception Assistée par Ordinateur.*

Cellule : *Module ou entité de base qui exécute une fonction. Il s'agit généralement d'une fonction logique telle que ET, OU, ... ou d'un élément de mémorisation tel qu'un flip-flop.*

CLB *(Configurable Logic Bloc): Module de logique programmable utilisé principalement dans les FPGA.*

CMOS : Complementary-Metal-Oxyde-Semiconductor.

CORDIC : *Coordinate Rotation Digital Computer.*

CPLD : PLD à technologie FLASH.

DFT : *Transformé de Fourier Discrète.*

DOD : département de la défense des Etats-Unis.

E2PROM*(Electricaly Erasable PROM) : PROM effaçable électriquement.*

EPLD *(Erasable Programmable Logic Devices): PAL à base de CMOS effaçables aux ultraviolets.*

FFT*(Fast Fourier Transformer) : La transformé de Fourier Rapide.*

FLEX : *Circuit programmable proposé par Altera.*

Flip-Flop: *Elément mémoire qui stocke les données en entrée au franc montant (ou descendant) de l'horloge, par opposition au latch.*

FPGA *(Field Programmable Gate Array): Circuit intégré déjà fabriqué mais qui peut être programmé pour effectuer la fonction désirée.*

FSM *(Finite State Machine): Machine d'états finis.*

GAL *(Genirec Array Logic): Réseau Logique Générique.*

IEEE *(Institute of Electrical and Electronics Engineers): Comité regroupant différents experts en électronique, qui émet un certain nombre de normes (*www.IEEE.org).

Instance : *Nom d'une cellule ou d'un module utilisé dans le circuit. Par exemple, les cellules sont celles d'une librairie et les instances représentent l'utilisation de ces cellules. Ainsi, si un circuit comporte uniquement deux flip-flops, celui-ci est composé d'une cellule mais de deux instance (reg1 et reg2).*

IOB : *Input Output Block.*

JTAG *(Joint Test Acces Group): Spécification de circuit permettant de tester le contenu et l'environnement d'un circuit, une fois celui-ci soudé sur un circuit imprimé.*

LCA *(Logic cell array): Réseau de cellules logiques.*

Librairie : *Regroupement de cellules, sous forme de symboles, dans une seule structure. Chaque fondeur fournit notamment une librairie de cellules utilisables pour la conception.*

LUTs *(Look Up Table) : Contiennent l'équivalent d'une table de transposition. Ces mémoire sont en pratique des SRAM.*

MOS : *Metal-Oxyde-Semiconductor.*

Netlist: *Fichier qui contient l'interconnexion de toutes les cellules constituant le circuit intégré. Permet de faire un placement-routage.*

PAL : *Programmable Array logic.*

PLD *(Programmable Logic Device) : Circuits logiques programmables.*

PROM : *Programmable Read Only Memory.*

RAM *(Random Access Memory): Mémoire vive.*

RTL *(Registre Transfer Level): Notation pour les modules VHDL décrits à haute abstraction.*

SRAM : *Static Random AccessMemory.*

VHDL *(VHSIC Hardware Description Langage) : Langage de description du matériel.*

VHSIC*(Very High Speed Integrated Circuit) : Circuit intégré à haute vitesse.*

Bibliographie

[1] M.Bellanger "Traitement numérique du signal"
Masson 1987

[2] A.Deluzurieux \ M.Rami "Cours d'électronique numérique et échantillonnée"
Eyrolles 1994

[3] M.Djeddi, L.Herous, S.Saad "Eléments de base de théorie et du traitement du signal"
Office des Publications Universitaires 1993

[4] A.Banerjee, A.Sundar Dhar, S.Banerjee"FPGA realization of a CORDIC based FFT processor for biomedical signal processing"
Elsevier, Microprocessors and Microsystems February 2001

[5] Christian Tavernier "Circuits logiques programmables"
Dunod 1996

[6] L.Dutrieux, D.Demigny "Logiques Programmables,architecture des FPGA et CPLD, méthodes de conception et langage VHDL".
Eyrolles 1997.

[7] M.aumiaux."initiation au langage VHDL".
Dunod 1999.

[8] M.meaudre, J.Weber "le langage VHDL".
Dunod 2001.

[9] Dominique Houzet "conception de circuits en VHDL, principes et methodologie".
Cepadues 2000.

[10] U.Heinkel, M.Padeffke, W.Haas, T.Buerner, H.Braisz, T.Gentner, A.Grassmann"The VHDL Reference".
John Wiley & sons, LTD 2000.

[11] Peter J.Ashenden"The designer's guide to VHDL". Morgan Kaufmann Publishers 2002.

[12] Thierry Schneider"VHDL: Méthodologie de design et techniques avancées ".
Dunod 2001.

[13] site:"http://www.andraka.com/files/crdcsrvy.pdf

[14] Documentation du CD de XILINX FOUNDATION.