REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE

Département d'Electronique

Projet de Fin D'Etudes

En vue de l'obtention du diplôme
d'Ingénieur d'Etat en Electronique

Thème

Architectures et Implémentations sur FPGA de l'opérateur CORDIC pour les communications sans fils

Proposé et dirigé par :
Mr Taghi Mohamed Oussaid
Mr Abdellouel Lahcen

Réalisé par :
Ben mesbah Wahiba
Tiouidiouine Zohra

الملخص

هذا العمل هو دراسة فنية لخوارزمية CIDROC (تنسيق الحسابات الرقمية بالدوران) لبناه الثلاث (المتسلسلة وبالتوازي مع وخط الأنابيب) واستخدامها في الاتصالات اللاسلكية، والقيام
أيضا بتمثيلها في برنامجLDHV وتنفيذه على بطاقةAGPF لحساب معظم الوظائف على أساس علم المثلثات

Résumé

Ce travail consiste à une étude technique de l'algorithme CORDIC (Coordinate Rotation Digital Computing) pour ces trois architectures (série, parallèle et pipeline) et leur utilisation dans les communications sans fils, et aussi faire une simulation en VHDL puis une implémentation sur une carte FPGA

Mots clés : CORDIC, communication sans fils, FPGA, VHDL

Abstract

This work is a technical study of the CORDIC algorithm (Coordinate Rotation Digital Computing) for these three architectures (serial, parallel and pipeline) and their use in the wireless communications, and also does a simulation in VHDL and implementation on a FPGA board.

Key world: CORDIC, Wireless communication, FPGA, VHDL,

Remerciements

On tient à exprimer nos vifs remerciements pour nos promoteurs M. TAGHI MOHAMED OUSSAID et M. ABDELLOUEL LAHCEN pour avoir proposé ce sujet, prodigué leurs précieux conseils tout au long de notre travail, leurs aides et leurs confiances.

On remercie très chaleureusement les membres de jury pour l'honneur qu'ils nous ont fait en acceptant d'être les rapporteurs de ce mémoire.

On remercie aussi tous nos enseignants de l'Ecole Nationale Polytechnique d'Alger, pour le savoir qu'ils nous ont transmis. Leur disponibilité et leur gentillesse.

On tient à remercier tous ceux qui ont participé de prés ou de loin à la réalisation de ce modeste travail.

Enfin, on aimerait adresser nos plus fervents remerciements a nos parents qui sans eux nous n'aurions pas atteint ce stade.

Sommaire

Introduction Générale 8
Chapitre1 : Les Communications sans fils 9
Introduction 9
1.2 Systèmes Sans fil Actuels 9
1.2.1 Systèmes de Téléphonie cellulaire 9
1.2.2 LANs sans fil 9
1.2.3 Wide Area Wireless Data Services (données sans fil des services à larges zones) 10
1.2.4 Réseaux Satellites 10
1.2.5 Radios Low-Cost Low-Power: Bluetooth et Zig bée 10
1.2.6 Radios d'Ultra large bande 10
1.3 Organisation fonctionnelle de la couche physique d'un système de communications sans fils 11
1.4 Les canaux Hertziens (canaux sans fils) : 12
1.5 Évolution des standards de communication 13
1.5.1 Bluetooth (IEEE 802.15.1) 14
1.5.2 WiFi (IEEE 802.11x) 14
1.5.3 WiMax (IEEE 802.16) 14
1.6 Utilisation de l'opérateur CORDIC dans les systèmes de communication sans fil : 15
1.6.1 Synthèse numérique directe de fréquence : 15
1.6.2 Modulation numérique : 15
1.6.3 Les systèmes OFDM et la FFT : 16
1.6.3 D'autres applications de communication : 17
Chapitre 2 : L'Algorithme CORDIC 18
2.1. Introduction : 18
2.2 Principe de l'algorithme CORDIC : 18
2.3 Les itérations fondamentales de l'algorithme CORDIC : 22
2.4 Etude de la précision de l'algorithme CORDIC 26
2.4.1Estimation de la précision CORDIC 27
Chapitre 3 : Architectures de l'Opérateur CORDIC 30
3.1 Introduction 30
3.2 CORDIC élémentaire 30
3.3 L'additionneur 31
3.3.1 Additionneur complet: 31

3.3.2 Additionneur par propagation de retenue 32
3.4 Soustracteur 32
3.5 Registre à décalage 33
3.6 Architecture série de l'algorithme CORDIC 34
3.6.1Machine d'état 35
3.7 Architecture parallèle 38
3.8 Architecture pipeline de CORDIC : 39
Chapitre 4 : Circuits FPGA 41
4.1 Introduction 41
4.2 Architecture des FPGA 42
4.2.1 Description des composants FPGA 42
4.2.2 Les CLB (configurable logic bloc) 43
4.2.3 Les IOB (input output bloc) 44
4.2.3.2 La configuration en sortie 46
4.2.4 Les interconnexions 46
4.2.4.1 Les interconnexions à usage général 46
4.2.4.2 Les interconnexions directes 47
4.2.4.3 Les longues lignes 48
4.4 Flot de conception 48
4.4.1 Vue d'ensemble du flot de conception 50
4.4.1.1 Description : code HDL : 50
4.4.1.2 Simulation fonctionnelle d'un modèle VHDL 51
4.4.1.3 La synthèse : 51
4.4.1.4 L'implémentation 51
Chapitre 5: Implémentations sur FPGA de l'opérateur CORDIC 52
5.1 Introduction 52
5.2 Manipulation de l'outil ModelSim 52
5.2.1 La simulation 52
5.3 Modélisation VHDL des architectures CORDIC 53
5.3.1 Additionneur/soustracteur 53
5.3.1.1 Code VHDL 53
5.3.1.2 La simulation de l'additionneur/soustracteur 54
5.3.2 registre à décalage 54

5.3.2.1 Code VHDL 55
5.3.2.2 la simulation de registre à décalage 57
5.3.3 La simulation de l'architecture Pipeline 57
5.3.4 Exemple de simulation de l'architecture série 58
5.3.5 Exemple de simulation de l'architecture parallèle 59

Conclusion générale: 60
ANNEXES 60
Annexe A 60
Annexe B 60
Annexe C 61
Bibliographie: 71

LISTE DES TABLEAUX ET DES FIGURES

Figure.1.1: Représentation fonctionnelle d'un système de transmission sans fils 11

Figure 1.2 Le schéma de principe d'une liaison hertzienne 12

Figure 1.3 schéma bloc d'un synthétiseur numérique direct (DDS) 15

Figure 1.4 schéma bloc d'un modulateur numérique 16

Figure 1.5 Schéma bloc d'un système OFDM 17

Figure 2.2: vecteur V d'amplitude r et d'argument θ 19

Table 2.1: Pour un hardware CORDIC de 8-bit 20

Tableau 2.2 : les valeurs approximatives de la fonction $\boldsymbol{\alpha i}$=arctan ($\mathbf{2} - \boldsymbol{i}$), en degré 23

Tableau 2.3 : Le choix des signes de la rotation des angles pour forcer z à zéro 24

Figure 2.5: les trois premières itérations parmi les 10 principales dans la rotation par 30°, ... 25

Figure 3.5 La précision numérique du processeur CORDIC 27

Figure 3.6 comparaison entre la valeur exacte de sinus thêta et les résultats de CORDIC pour différents nombres des itérations (thêta = 30°) 28

Tableau 2.4 les résultats de simulation de CORDIC pour nombre de bits variables 28

Figure 3.7 comparaison entre la valeur exacte de cosinus thêta et les résultats de CORDIC pour différents nombres de bits (thêta = 30°) 29

Figure 3.8 L'erreur produit par l'algorithme CORDIC pour nombre de bits variable..... 29

Figure 3 : CORDIC élémentaire..... 30

Figure 3.1 Schéma à partir des formules d'un additionneur complet..... 32

Figure 3.3 Registre a décalage SISO ou SIPO..... 34

Figure 3.4 Architecture série de l'algorithme CORDIC..... 35

Figure 3.5 : Représentation des signaux d'entée/sortie de la machine d'état..... 36

Figure 3.6 : machine d'état de l'architecture série de l'algorithme CORDIC..... 37

Figure 3.5 L'architecture parallèle de l'algorithme CORDIC..... 38

Figure 3.4 L'architecture pipline de l'algorithme CORDIC..... 39

Figure.4.1 : Architecture interne d'un FPGA..... 42

Figure4.2 : Cellules logiques (CLB)..... 43

Figure 4.3: Input Output Block (IOB)..... 45

Figure4.4 : Connexions à usage général et matrice de commutation..... 47

Figure4.5 : Les interconnexions directes..... 48

Figure4.6: Les longues lignes..... 48

Tableau4.1 : Flots de conception des fabricants de FPGA..... 49

Figure 4-9 - flot de conception d'un circuit numérique..... 50

Figure 5.1 : la simulation d'un additionneur/soustracteur..... 54

Figure 5.2 : la simulation d'un registre à décalage..... 55

Figure5.3 Simulation de l'algorithme CORDIC ou les outputs sont Sin et Cos..... 56

Tableau 5.1: Les outputs Sin / Cos pour certains angles..... 56

Figure 5.2 Dataflow de l'architecture pipeline..... 57

Figure5.4 Simulation de l'algorithme CORDIC ou les outputs sont Sin et Cos…… 58

Introduction Générale

On peut s'attendre à ce que très bientôt, tous les appareils qui servent à générer ou à sauvegarder des données comportent des fonctions intégrées de transmission sans fil ou puissent être reliés à des réseaux sans fil. Ainsi, les téléphones cellulaires et les assistants numériques sont de plus en plus perfectionnés, et sont souvent dotés de plusieurs technologies sans fil.

Ces technologies comportent des économies et des gains d'efficacité indéniables, l'opérateur CORDIC (COordinate Rotation Digital Computer) a beaucoup contribué dans ce sens du fait de sa simplicité et son efficacité pour le calcul d'une gamme de fonctions complexes très utilisées dans la chaîne de communication. S'appuyant sur une technique décalage-addition et de rotation vectorielle, l'algorithme CORDIC calcule par approximation la plupart des fonctions trigonométriques ou apparentées telles que sinus, cosinus, arc tangente et racine carré, etc... . Les systèmes de communication numériques l'utilisent également afin de produire rapidement des conversions polaires-cartésiennes sur des signaux échantillonnés.

Le présent travail, qui rentre dans le cadre de notre projet de fin d'études, vise justement à faire toute la lumière sur l'opérateur CORDIC, ses fondement mathématiques, ses variantes algorithmiques ainsi que les architectures correspondantes.

Ce rapport se compose de cinq chapitres.

Dans le Chapitre 1, nous faisons une brève présentation des communications sans fils où nous illustrons l'importance de la place qu'occupe l'opérateur CORDIC dans ce genre de systèmes. Au Chapitre 2, une étude détaillée de l'algorithme CORDIC est élaborée exposant ses bases mathématiques et toutes ses variantes.

Le Chapitre 3, présente les trois principales architectures de l'opérateur CORDIC à savoir l'architecture série, parallèle et l'architecture pipeline, leurs avantages et leurs inconvénients.

Dans le Chapitre 4, nous étudions les circuits FPGA, leur architecture, ainsi que leur flôt de conception.

Le Chapitre 5 est consacré à l'implémentation de l'opérateur sur la carte FPGA, où nous présentons et expliquons la modélisation VHDL des 3 différentes architectures de CORDIC ainsi que les résultats de simulations sur Modelsim et Xilinx Navigator.

Enfin, nous clôturons avec une conclusion générale pour résumer les résultats obtenus ainsi que les perspectives de poursuite de ce travail.

Chapitre1 : Les Communications sans fils

Introduction

Les communications sans fil sont le segment de croissance le plus rapide de l'industrie de communications. En tant que tel, il a capté l'attention des média et l'imagination du public. Les systèmes cellulaires ont connu la croissance exponentielle pendant la décade dernière et il y a actuellement environ deux milliards d'utilisateurs dans le monde entier. Effectivement, les téléphones cellulaires sont devenus un outil d'affaires critique et une partie de vie quotidienne dans les pays les plus développés et supplantent rapidement des systèmes filaires archaïques dans beaucoup de pays en développement. En plus, les réseaux local sans fil complètent actuellement ou remplacent des réseaux connectés dans beaucoup de maisons, entreprises et campus. Beaucoup de nouvelles applications, en incluant des réseaux de détecteur sans fil, ont automatisé des autoroutes et des usines, des maisons intelligentes et des appareils, et télémédecine à distance, émergent des idées de recherche aux systèmes concrets. La croissance explosive de systèmes sans fil couplés avec la prolifération de portables et d'ordinateurs portables indique un avenir brillant pour les réseaux sans fil, tant comme seul de l'éventaire les systèmes que dans le cadre de la plus grande infrastructure de mise en réseau. Cependant, beaucoup de défis techniques restent dans la conception des réseaux sans fil robustes qui livrent la performance nécessaire pour soutenir des applications émergeant.

1.2 Systèmes Sans fil Actuels

1.2.1 Systèmes de Téléphonie cellulaire

La téléphonie mobile, ou téléphonie cellulaire est un moyen de télécommunication par téléphone sans fil. Ce moyen de communication s'est largement répandu à la fin des années 1990. Les systèmes de téléphone cellulaire sont extrêmement populaires et lucratifs dans le monde entier : ceux-ci sont les systèmes qui ont enflammé la révolution sans fil. Les systèmes cellulaires fournissent à la voix bilatérale et à la communication de données avec la couverture régionale, nationale, ou internationale.

1.2.2 LANs sans fil

Les LANs sans fil fournissent des données de grande vitesse dans une petite région, par exemple un campus ou un petit bâtiment, comme le mouvement d'utilisateurs d'un endroit à l'autre. Les appareils sans fil qui accèdent à ces LANs sont typiquement stationnaires ou mobiles aux vitesses piétonnières.

1.2.3 Wide Area Wireless Data Services (données sans fil des services à larges zones)

Ce type de systèmes a fourni des données sans fil aux utilisateurs de la haute mobilité sur une très grande région de couverture. Dans ces systèmes une région géographique donnée est assurée l'entretien par les stations basées montées sur les tours, les toits, ou les montagnes. Les stations basées peuvent être raccordées au réseau connecté d'une colonne vertébrale ou former un multi bond le réseau sans fil ad hoc.

1.2.4 Réseaux Satellites

Les systèmes satellites commerciaux sont une autre composante importante des communications sans fil l'infrastructure. Les systèmes de Géosynchrones incluent Inmarsat (*International maritime satellite organization)* et OmniTRACS. Celui-là est destiné principalement pour la transmission de voix analogique des endroits lointains. Par exemple, il est communément utilisé par les journalistes pour fournir des reportages vivants des zones de guerre.

1.2.5 Radios Low-Cost Low-Power: Bluetooth et Zig bée

Comme les radios diminuent alors le prix et la consommation d'énergie, il devient réalisable de les fixer dans plus de types d'appareils électroniques, qui peuvent être utilisés pour créer des maisons intelligentes, des réseaux de détecteur et d'autres applications fascinantes. Deux radios ont émergé pour soutenir cette tendance : Bluetooth et Zig bée.

1.2.6 Radios d'Ultra large bande

Les radios Ultra wide band (UWB) sont des radios extrêmement à large bande avec très hauts débits des données potentiels. Le concept de communications ultra wide band actuellement créé avec le transmetteur d'espace d'étincelle de Marconi, qui a occupé une très large bande passante. Cependant, comme seulement un utilisateur de taux faible simple pourrait occuper le spectre, les communications à large bande a été abandonné en faveur des techniques de communication plus efficaces.

1.3 Organisation fonctionnelle de la couche physique d'un système de communications sans fils

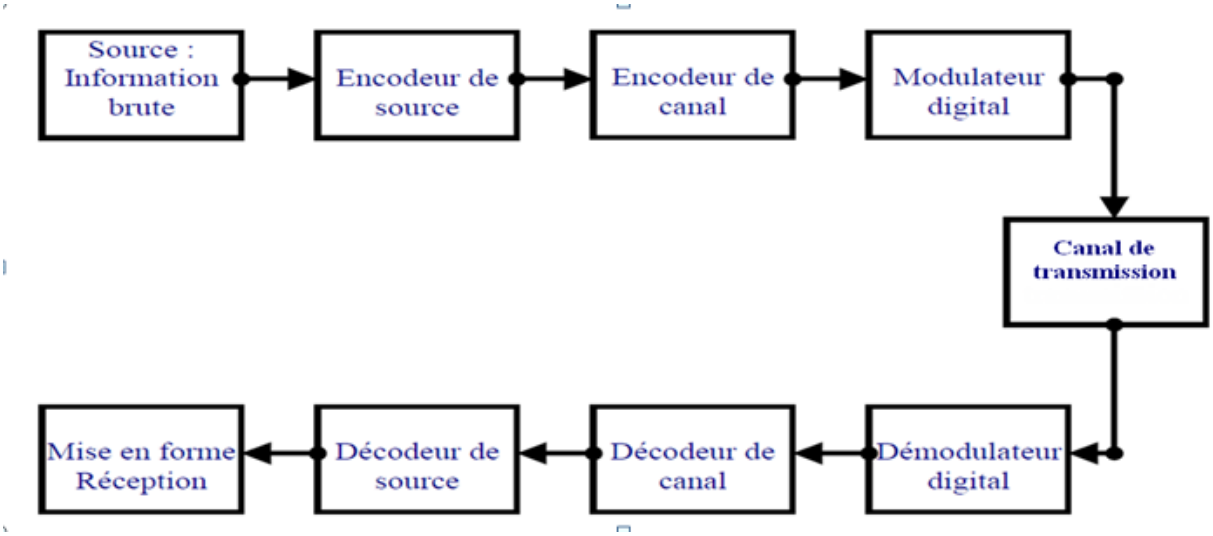

Figure.1.1: Représentation fonctionnelle d'un système de transmission sans fils

La figure 1.1 permet de présenter les fonctions de base d'un système de transmission numérique sans fils remplissant le rôle d'interface entre le signal continu adapté au médium de transmission et les informations binaires.

La source du message correspond aux données de l'émetteur, et réciproquement, la destination du message du destinataire se trouvant à l'autre extrémité de la ligne de transmission.

Le codeur source peut éventuellement modifier les éléments binaires à transmettre.

Ceci est par exemple utile pour appliquer des opérations de compression permettant de réduire le nombre de bits effectivement présents dans la séquence à envoyer vers le récepteur. Cette étape repose en grande partie sur une bonne connaissance des paramètres statistiques du signal transmis.

Le codage de canal permet entre autres d'adapter cette séquence de manière à lui offrir une meilleure résistance aux perturbations et autres bruits qui peuvent intervenir lors du transit sur le canal de transmission. Ce codage intervient sous la forme d'algorithmes à base de codes, comme les codes cycliques, le code de Hamming ou encore les turbo-codes.

1.4 Les canaux Hertziens (canaux sans fils) :

Le support de transmission est l'atmosphère, l'espace. Les informations sont transmises par des ondes électromagnétiques et la gamme des fréquences est très vaste puisque cela va du kHz jusqu'à 1016 Hz. Ces canaux Hertziens sont par nature soumis à de nombreuses perturbations. Leur avantage est de ne demander la création d'aucun fil ou câble de transmission.[4]

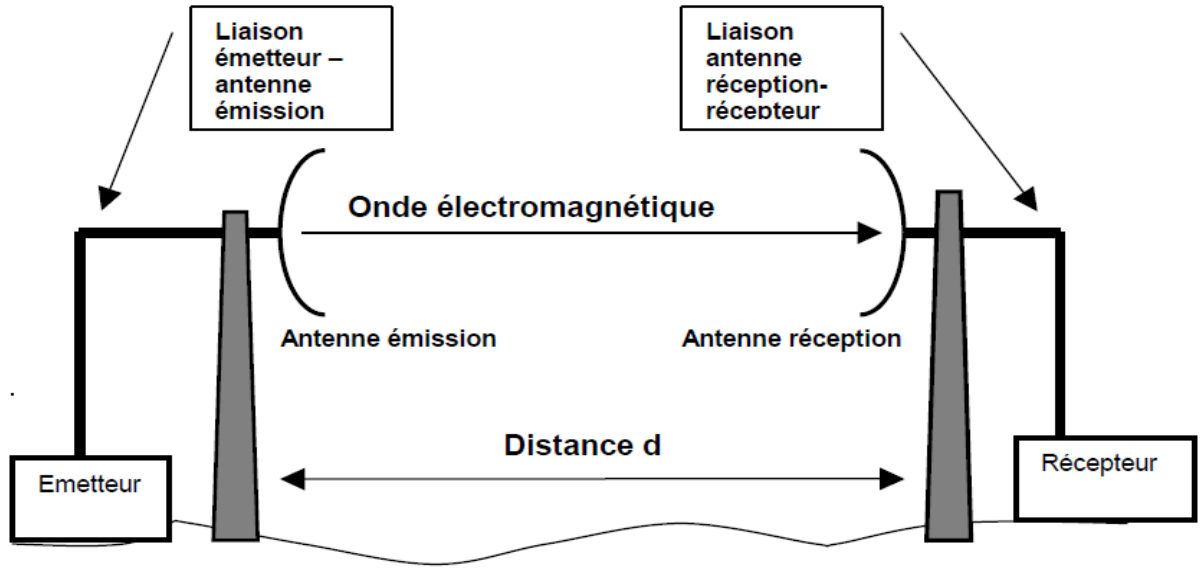

Figure 1.2 Le schéma de principe d'une liaison hertzienne

Emetteur : Il est caractérisé par sa puissance émise PE. Ici PE sera exprimée en dBm ou dBW. Ordre de grandeur : de quelques mW (0dBm) à plusieurs kW (> 30dBW).

Liaison émetteur- antenne émission : elle est généralement réalisée en câble coaxial. A plus haute fréquence (> quelques GHz), elle peut être réalisé en guide d'onde (voir cours de physique). Elle est caractérisée par son atténuation LE, exprimée en dB.

Dans les petits systèmes, où tout est intégré (WiFi, téléphone mobile, etc..) cette liaison n'existe pas (LE = 0dB)

Antenne émission : Elle est caractérisée par son Gain d'antenne GE, exprimé en dB. Distance d : c'est la distance entre l'émetteur et le récepteur. On peut montrer (à partir du calcul de la sphère de l'antenne isotrope et de la définition du gain d'antenne), que la distance entre

l'émetteur et le récepteur, introduit une atténuation AEL (pour atténuation en espace libre) égale à $A_{EL}=20\log(4.\Pi .d/\lambda)$.

Cette grandeur est exprimée en dB

Liaison antenne réception- récepteur : comme la liaison émetteur-antenne émission, la liaison antenne réception-récepteur est caractérisée par l'atténuation L_R, exprimée en dB.

Antenne réception : Elle est caractérisée par son gain d'antenne G_R, exprimé en dB.

Récepteur : Le paramètre qui nous intéresse ici est P_R, puissance reçue par le récepteur.

Elle est généralement exprimée en dBm.

1.5 Évolution des standards de communication

Il convient de constater que le développement des techniques nécessaires à la mise en place des systèmes sans fil et radio mobiles se sont largement développées. En effet, la réception hétérodyne a été inventée dans les années dix et les techniques de traitement du signal telles que l'étalement du spectre ou les modulations numériques ont été développées dans les années quarante. Cependant, depuis les premières expériences de Marconi à la fin du XIX e siècle, les communications sans fil entre deux éléments mobiles ont été limitées à des applications professionnelles et ce, jusqu'à la fin des années soixante dix. Il a fallu attendre les années quatre vingt avant de voir la naissance des systèmes de téléphonie mobile cellulaire, de première génération, entièrement analogiques comme le NMT (Nordic *Mobile Telephony*), l'AMPS (Advanced *Mobile Phone Service)*, le TACS (Total *Access Cellular System*) ou le Radiocom2000. Néanmoins, les ressources spectrales réduites, la courte durée de vie des batteries et le coût du terminal, ont été des obstacles qui ont restreint les débuts du développement des téléphones portables de première génération.

La seconde génération des systèmes cellulaires est apparue au début des années 90 avec le DECT (Digital *Enhanced Cordless Telecommunications),* le GSM (*Global System for Mobile communications,* Europe), le PDC (Japon), et le PCS (États-Unis). Prévue initialement pour des applications de transport de la parole et de données à faibles débits (9600 bits/s) autour des fréquences (935-960 MHz) ou (890-915 MHz), cette norme a permis l'essor des communications mobiles. Les performances (efficacité spectrale, sécurité, fonctionnalités) de la seconde génération de systèmes cellulaires sont plus importantes que celles de la première génération.

De ce fait, on peut noter la naissance des principaux standards suivants :

1.5.1 Bluetooth (IEEE 802.15.1)

Cette norme fonctionne dans la bande ISM (Industrial *Scientific Medical)* entre 2.4 GHz et 2.48 GHz utilisable librement (sans Licence ni autorisation). Elle permet des débits pouvant

atteindre 3 Mbits/s dans un rayon de moins de 100 mètres. La technologie Bluetooth est également de plus en plus utilisée dans les téléphones portables, afin de leur permettre de communiquer avec des ordinateurs ou des assistants personnels et surtout avec des dispositifs mains-libres tels que des oreillettes Bluetooth. *ZigBee (802.15.4)*

C'est une norme caractérisée par un faible débit et un faible coût avec pour atout majeur une faible consommation. Elle fonctionne dans la bande de fréquence située autour de 2.4 GHz.

1.5.2 WiFi (IEEE 802.11x)

C'est un ensemble de normes concernant les réseaux locaux sans fil (WLAN) qui offrent des débits pouvant atteindre théoriquement 54 Mbits/s (de 11 Mbit/s en 802.11b à 54 Mbit/s en 802.11a/g) sur une bande de fréquence de 2.4 GHz (802.11b/g) et 5 GHz (802.11a). Son équivalent en Europe est le Hiper LAN.

1.5.3 WiMax (IEEE 802.16)

Ce standard permet un débit théorique de 70 Mbits/s sur un rayon de 50 km maximum. La norme WiMax Mobile devrait permettre des services comme la communication en Voie IP (Téléphonie sur Réseau IP) sur téléphone portable ou encore l'accès à des services mobiles en hauts débits. Son équivalent en Europe est le HiperMAN.

Ainsi, le développement rapide des communications sans fil et l'émergence des nouveaux standards sollicite la convergence vers la quatrième génération de communications mobiles, initialement prévue pour les années 2010, semblerait présenter ses toutes premières réalisations commerciales légèrement plus tôt. Cette avance par rapport aux prévisions est notamment due au fait que la quatrième génération ne sera pas l'issue d'une révolution dans les communications (comme l'ont été les seconde et troisième générations), mais plutôt la convergence de différents standards, applications et produits. En effet, il semble pertinent de considérer la 4G comme la convergence des standards et des technologies couverts par la 3G et les réseaux locaux sans fil (WLAN). Le but de la "4G" est d'améliorer les performances de la troisième génération, sans changer fondamentalement le contenu ni les applications prévues au départ pour la 3G. [3]

1.6 Utilisation de l'opérateur CORDIC dans les systèmes de communication sans fil :

L'opérateur CORDIC peut être employé pour l'exécution efficace de divers modules fonctionnels dans un système de communication numérique. La plupart des applications de CORDIC dans les communications emploient le système du même rang circulaire dans un ou les deux modes CORDIC d'opération. Le RM-CORDIC (mode rotation) est principalement employé pour produire des messages mélangés, alors que le VM-CORDIC (mode vecteur) est principalement employé pour estimer des paramètres de phase et de fréquence. Nous décrivons brièvement ici certaines applications de communication importantes utilisant le CORDIC.

1.6.1 Synthèse numérique directe de fréquence :

La synthèse numérique directe est le processus de produire des formes d'onde sinusoïdales directement dans le domaine numérique. Un synthétiseur numérique direct (DDS) se compose d'un accumulateur de phase et d'un convertisseur de phase-à-forme d'onde. Le circuit de phase-generator incrémente la phase où la fréquence porteuse normale dans chaque cycle et alimente l'information de phase au convertisseur de phase-à-forme d'onde. Le convertisseur de phase-à-forme d'onde pourrait être réalisé par un RM-CORDIC, Les formes d'onde de cosinus et de sinus sont obtenues, respectivement, par les sorties de CORDIC. [6]

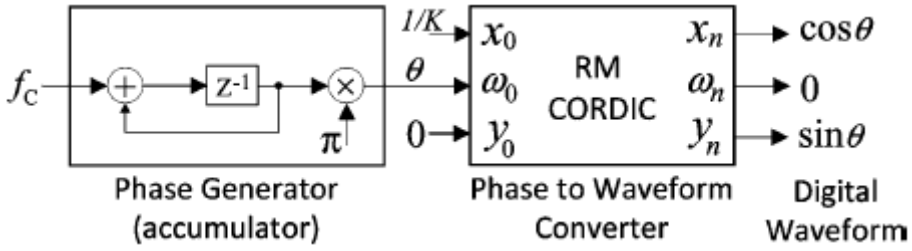

Figure 1.3 schéma bloc d'un synthétiseur numérique direct (DDS)

1.6.2 Modulation numérique :

La figure présente le schéma générale d'un modulateur numérique, La principale difficulté dans la conception d'un modulateur numérique se trouve dans la partie de génération numérique de fréquence, qui a pour rôle de calculer les échantillons de sinusoïde. Il faut donc

limiter autant que possible le nombre de fonctions arithmétiques complexes intervenant dans la chaîne de génération d'échantillons, ou transformer ces opérations en sous-éléments plus simples pouvant être calculés en parallèle. L'algorithme CORDIC permet de calculer facilement les valeurs des échantillons d'une sinusoïde sans utiliser de multiplication ni de boucle de rétroaction complexe [6]

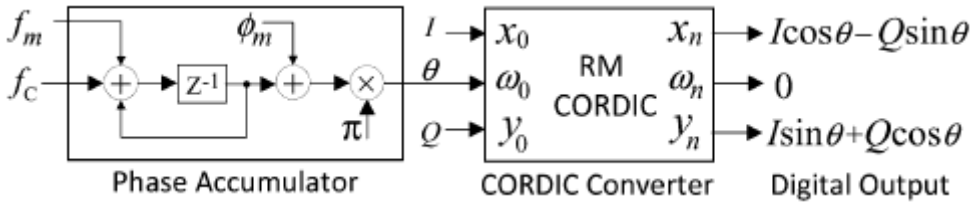

Figure 1.4 schéma bloc d'un modulateur numérique

1.6.3 Les systèmes OFDM et la FFT :

OFDM (Orthogonal frequency-division multiplexing) procédé de codage de signaux numériques par répartition en fréquences orthogonales sous forme de multiples sous-porteuses.

L'operateur CORDIC est un élément essentiel du bloc de canal d'estimation et de compensation où l'estimation de décalage de la porteuse implique le calcul des angles de phase des nombres complexes et la compensation de la fréquence de porteuse exige génération d'exposants de phase.

En plus l'opérateur CORDIC intervient dans les blocs de FFT (Fast Fourier Transform) et FFT inverse pour la même raison.

Rappelons que l''équivalent passe-bas d'un signal OFDM est exprimé ainsi :

$$\nu(t) = \sum_{k=0}^{N-1} I_k e^{i2\pi kt/T}, \quad 0 \le t < T,$$

Où $\{I_k\}$ sont les symboles de donnée, N est le nombre de sous-porteuses et T la durée du bloc OFDM. L'espacement entre porteuses de $1/T$ Hz rend les sous-porteuses orthogonales entre elles. La figure 1.5 présente Le schéma fonctionnel d'un système OFDM,

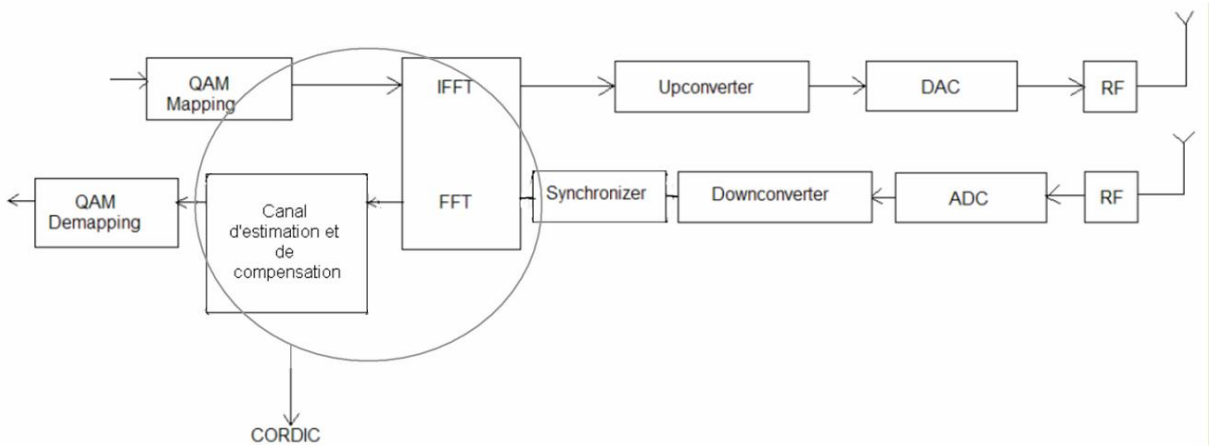

Figure 1.5 Schéma bloc d'un système OFDM

1.6.3 D'autres applications de communication :

Le calcul d'angle dans la VM CORDIC peut être employé pour détecter des signaux de FM et de FSK et pour estimer les paramètres de phase et de fréquence. Un VM-CORDIC simple peut être employé pour exécuter ces calculs pour l'exécution d'une trancheuse pour une constellation d'ordre élevé comme le 32-APSK utilisé dans DVB-S2. Des circuits CORDIC fonctionnant en les deux modes sont également exigés dans les récepteurs numériques pour que l'étape de synchronisation effectue une évaluation de phase ou de fréquence suivie d'une étape de correction.

Chapitre 2 : L'Algorithme CORDIC

2.1. Introduction :

L'algorithme CORDIC, (acronyme de COordinate Rotation DIgital Computing) a été créé à l'origine par J.E.Volder en 1959 pour effectuer des calculs tels que les rotations de vecteurs ou les changements de coordonnées cartésiennes-polaires et polaires-cartésiennes dans le plan euclidien, dans des applications de navigation aérienne. C'est une méthode simple et efficace pour le calcul d'une gamme de fonctions complexes qui s'appuie sur une technique d'additions et de décalage entre vecteurs. L'algorithme calcule par approximation la plupart des fonctions basées sur la trigonométrie. Il exécute des rotations sans utiliser d'opérations de multiplication. Un autre avantage de cet algorithme est qu'il permet d'obtenir une précision déterminée à l'avance en effectuant un nombre d'itération donné .

2.2 Principe de l'algorithme CORDIC :

L'algorithme CORDIC est basé sur des calculs des fonctions trigonométriques, son principe est d'effectuer des rotations sur un vecteur de base pour un angle donné. Si un vecteur V avec des coordonnées (x, y) est tourné par un angle Φ alors qu'un nouveau vecteur V' (x',y') où x' et y' peuvent être obtenus utilisant x, y et Φ par la méthode suivante.

$$X = r\cos\theta,\ Y = r\sin\theta \qquad \textbf{(2.1)}$$

$$V' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x.\cos(\Phi) - y.\sin(\Phi) \\ y.\cos(\Phi) + x.\sin(\Phi) \end{pmatrix} \qquad \textbf{(2.2)}$$

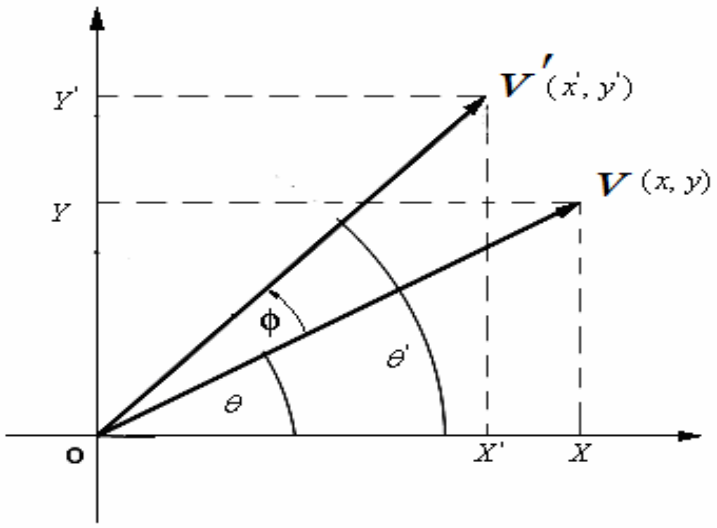

Figure 2.1: Rotation d'un vecteur V par un angle Φ

Dans la figure 2.1 le vecteur V (x, y) peut être décomposé en deux composantes suivant l'axe des x r cosθ et suivant l'axe des y r sinθ.

La figure 2.2 montre la rotation de vecteur V=$\begin{pmatrix} x \\ y \end{pmatrix}$ par l'angle Φ.

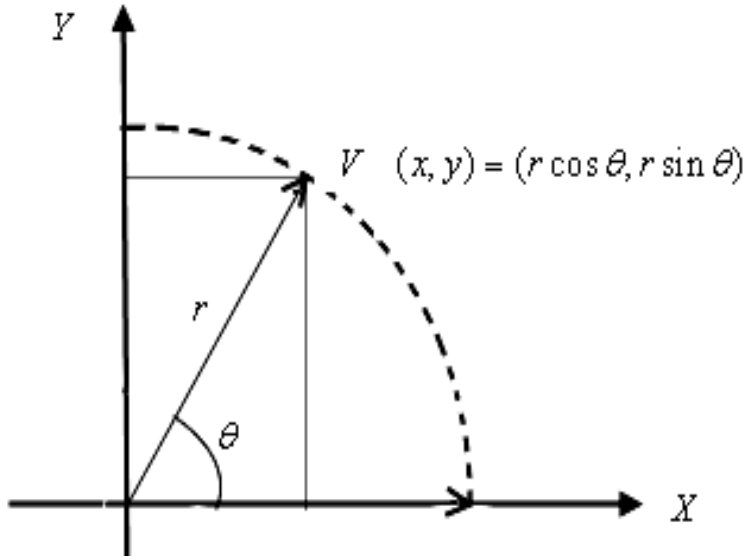

Figure 2.2: vecteur V d'amplitude r et d'argument θ

i.e. $\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$ **(2.3)**

Les deux vecteur V et V′ ont leurs amplitudes et arguments (r, θ) et (r,θ′) respectivement où V′ est la rotation de vecteur V par un angle Φ dans le sens inverse des aiguilles d'une montre. De la figure 2.1 on voit que

$\theta' - \theta = \Phi$ **(2.4)**

i.e. $\theta' = \theta + \Phi$ **(2.5)**

$$OX' = x' = r\cos\theta'$$
$$= r\cos(\theta + \phi)$$
$$= r(\cos\theta.\cos\phi - \sin\theta.\sin\phi)$$
$$= (r.\cos\theta)\cos\phi - (r.\sin\theta)\sin\phi \quad \textbf{(2.6)}$$

Utilisant la figure 2.2 et l'équation 2.3 OX' peut être représenté comme suit :

$OX' = x' = x\cos\phi - y\sin\phi$ **(2.7)**

La même chose pour OY'

$OY' = y' = y\cos\phi + x\sin\phi$ **(2.8)**

De la même façon si on fait tourner le vecteur V d'un angle ϕ dans le sens des aiguilles d'une montre, on obtient les équations suivantes :

$x' = x\cos\phi + y\sin\phi$ **(2.9)**

$y' = x\sin\phi - y\cos\phi$ **(2.10)**

Les équations (2.7), (2.8), (2.9), (2.10) peuvent être représentées dans la matrice suivante

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\Phi & \pm\sin\Phi \\ \pm\sin\Phi & \cos\Phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad \textbf{(2.11)}$$

Les équations séparées de *x*' et de *y*' peuvent être réécrite comme suit

$$x' = x.\cos(\phi) \pm y.\sin(\phi) \quad \textbf{(2.12)}$$

$$y' = y.\cos(\phi) \pm x.\sin(\phi) \quad \textbf{(2.13)}$$

On fait sortir cos Φ en facteur, ensuite on considère l'angle θ comme un ensemble de petits angles dont le tangent de chaque un est choisit comme une puissance de deux inverse.
Donc les équations peuvent être réécrites comme des formules itératives :

$$x' = \cos\phi\,(x \pm y\tan\phi) \quad \textbf{(2.14)}$$

$$y' = \cos\phi\,(y \pm x\tan\phi) \quad \textbf{(2.15)}$$

$z' = z \pm\phi$, ici ϕ est l'angle de rotation (le signe ± est la direction de rotation) et z est l'argument. Pour la facilité des calcules on prend d'abord la rotation dans le sens inverse des aiguilles d'une montre. Réarrangeant les équations (2.7) et (2.8).

$$x' = \cos(\phi)\,[x - y.\tan(\phi)] \quad \textbf{(2.16)}$$

$$y' = \cos(\phi)\,[y + x.\tan(\phi)] \quad \textbf{(2.17)}$$

Le tan (ϕ) est limité pour une valeur de 2^{-i}. Dans le digital hardware ceci indique une simple opération de décalage. En outre, si ces rotations sont exécutées itérativement et dans les deux sens chaque valeur de tan (Φ) est représentable. Avec $\phi = \arctan(2^{-i})$ le terme cosinus peut être également simplifié $\cos(\phi) = \cos(-\phi)$ c'est une constante pour un nombre fixe d'itérations. Maintenant cette rotation itérative peut être exprimée comme :

$$x_{i+1} = k_i[x_i - y_i\, d_i\, 2^{-i}] \quad \textbf{(2.18)}$$

$$y_{i+1} = k_i[y_i + x_i\, d_i\, 2^{-i}] \quad \textbf{(2.19)}$$

Où, i indique le nombre des rotations exigé pour atteindre l'angle demandé du vecteur demandé, k_i=cos (arctan(2^{-i})) et $d_i = \pm 1$. Le produit des k_i représente ce qu'on appelle le facteur K :

$$K = \prod_{i=0}^{n-1} k_i \quad \textbf{(2.20)}$$

Où, $\prod_{i=0}^{n-1} k_i = \cos\Phi_0 \cos\Phi_1 \cos\Phi_2 \cos\Phi_3 \ldots\ldots\ldots \cos\Phi_{n-1}$(Φ est l'angle de rotation ici pour n rotations).

Table 2.1: Pour un hardware CORDIC de 8-bit

i	$\tan\Phi_i$	$\Phi_i = \arctan(2^{-i})$	$\Phi_i$en radian
0	1	45°	0.7854
1	0.5	26.565°	0.4636
2	0.25	14.036°	0.2450
3	0.125	7.125°	0.1244
4	0.0625	3.576o°	0.0624

5	0.03125	1.7876°	0.0312
6	0.015625	0.8938°	0.0156
7	0.0078125	0.4469°	0.0078

K est le gain sa valeur change avec l'augmentation de nombre d'itération. Pour le hardware CORDIC de 8-bit la valeur approximative de K

$$K = \prod_{i=0}^{7} \cos\Phi_i = \cos\Phi_0 \cos\Phi_1 \cos\Phi_2 \cos\Phi_3 \cos\Phi_4 \cos\Phi_5 \cos\Phi_6 \cos\Phi_7$$
$$= \cos 45° . \cos 26.565° ..... \cos 0.4469°$$
$$= 0.6073 \qquad \textbf{(2.21)}$$

Du tableau ci-dessus on peut voir jusqu'à la précision 0.4469° pour le hardware CORDIC de 8- bits. Ces Φ_i sont enregistrés dans la ROM de hardware CORDIC comme un tableau de consultation (LUT look up table). Maintenant on prend l'exemple de balance on comprend comment l'algorithme CORDIC travaille.

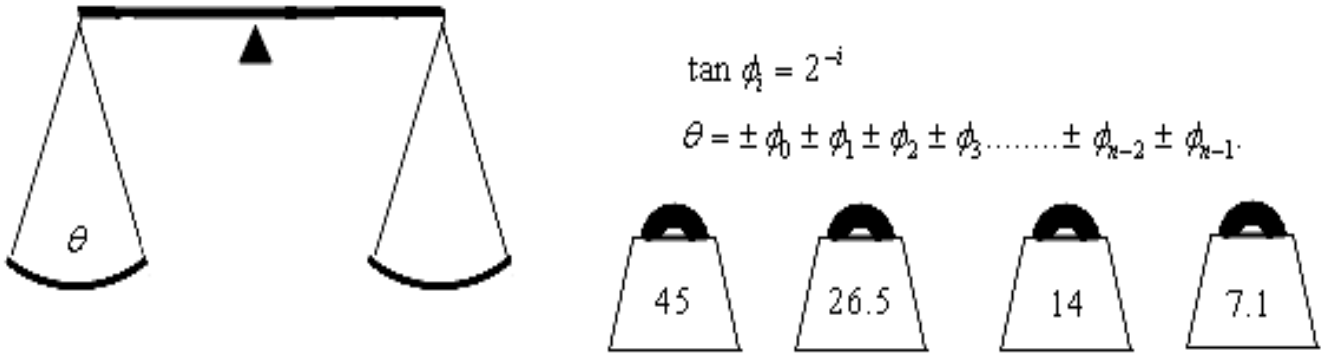

Figure 2.3: Une balance ayant θ sur un côté et des petits poids (angles) sur l'autre côté.

Dans la figure ci-dessus, tout d'abord, maintenir l'angle d'entrée θ sur le côté gauche de la balance et si la balance tourne autour en sens inverse des aiguilles d'une montre alors ajoute la valeur la plus élevée dans le tableau sur l'autre côté.

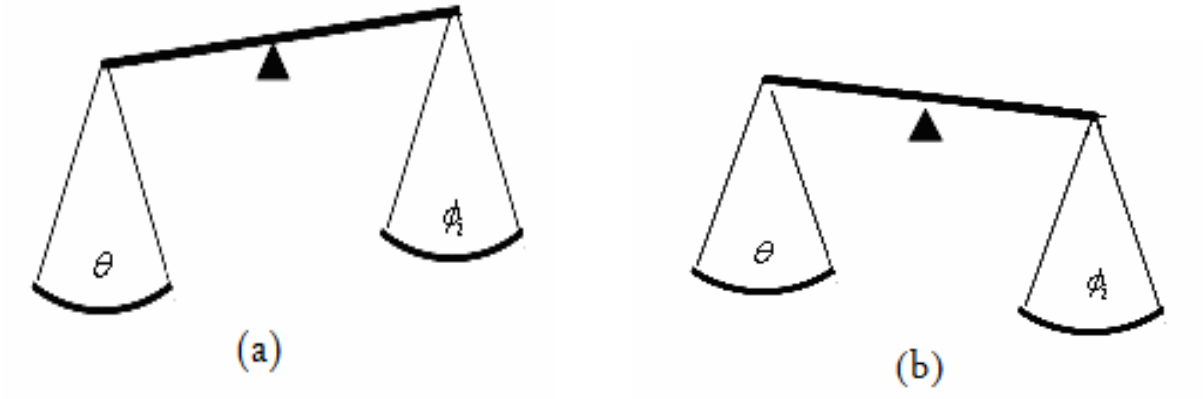

Figure 2.4: l'inclination de la balance due à la différence du poids des deux côté.

Donc, si la balance montre une inclination gauche comme sur le schéma 2.4 (a) puis d'autres poids sont exigés pour être ajouté dans le côté droit ou dans le terme de l'angle si le θ est plus grand que le total Φ_i donc ajouter des poids aussi possible pour atteindre θ mais dans l'autre main si la balance montre une inclination comme sur le schéma 2.4 (b) puis des poids exigés pour être retiré du côté droit ou dans le terme d'angle si le θ est moins que le total Φ_i le processus est répété aussi possible pour atteindre l'angle θ.
Une représentation matricielle de l'algorithme CORDIC pour un hardware de 8-bit :

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\ \Phi_i & \pm\sin\ \Phi_i \\ \pm\sin\ \Phi_i & \cos\ \Phi_i \end{pmatrix} \begin{pmatrix} xi \\ yi \end{pmatrix} \qquad \textbf{(2.22)}$$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos\ \Phi_0 & \pm\sin\ \Phi_0 \\ \pm\sin\ \Phi_0 & \cos\ \Phi_0 \end{pmatrix} \begin{pmatrix} \cos\ \Phi_1 & \pm\sin\ \Phi_1 \\ \pm\sin\ \Phi_1 & \cos\ \Phi_1 \end{pmatrix} \ldots\ldots$$
$$\ldots\ldots \begin{pmatrix} \cos\ \Phi_7 & \pm\sin\ \Phi_7 \\ \pm\sin\ \Phi_7 & \cos\ \Phi_7 \end{pmatrix} \begin{pmatrix} x0 \\ y0 \end{pmatrix} \qquad \textbf{(2.23)}$$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \cos\Phi_0\ \cos\Phi_1 \ldots\ldots\ldots \cos\Phi_7 \begin{pmatrix} 1 & \pm\tan\ \Phi_0 \\ \pm\tan\ \Phi_0 & 1 \end{pmatrix} \begin{pmatrix} 1 & \pm\tan\ \Phi_1 \\ \pm\tan\ \Phi_1 & 1 \end{pmatrix}$$

$$\ldots\ldots\ldots \begin{pmatrix} 1 & \pm\tan\ \Phi_7 \\ \pm\tan\ \Phi_7 & 1 \end{pmatrix} \begin{pmatrix} x0 \\ y0 \end{pmatrix} \qquad \textbf{(2.24)}$$

Ainsi, le facteur d'échelle = $\cos\Phi_0\ \cos\Phi_1 \ldots\ldots\ldots \cos\Phi_7$
= 0.6073
$= \frac{1}{1.6466}$ **(2.25)**

De l'équation (2.22) le cosinus et le sinus de l'angle θ peuvent être représenté sous la forme d'une matrice comme

$$\begin{pmatrix} cos\,\theta \\ sin\,\theta \end{pmatrix} = \begin{pmatrix} 1 & \pm\tan\ \Phi_0 \\ \pm\tan\ \Phi_0 & 1 \end{pmatrix} \begin{pmatrix} 1 & \pm\tan\ \Phi_1 \\ \pm\tan\ \Phi_1 & 1 \end{pmatrix} \ldots\ldots$$
$$\ldots\ldots \begin{pmatrix} 1 & \pm\tan\ \Phi_7 \\ \pm\tan\ \Phi_7 & 1 \end{pmatrix} \begin{pmatrix} 0.6073 \\ 0 \end{pmatrix} \qquad \textbf{(2.26)}$$

D'une manière générale le facteur d'échelle K=$\prod_1^n Ki = 1/\prod_1^n \sqrt{(1 + 2\text{^}(-2i)}$

2.3 Les itérations fondamentales de l'algorithme CORDIC :

Pour simplifier chaque rotation, choisissant α_i (l'angle de rotation de la iième itération) tel que α_i=$d_i\ 2^{-i}$. d_i prend la valeur +1 ou -1 selon la rotation donc

$$x_{i+1} = x_i \; - \; y_i \; d_i \; 2^{-i} \qquad \textbf{(2.27)}$$

$$y_{i+1} = y_i \; + x_i \; d_i \; 2^{-i} \qquad \textbf{(2.28)}$$

$$z_{i+1} = z_i \; - \; d_i \; tan^{-1} \; 2^{-i} \qquad \textbf{(2.29)}$$

Le calcul de x_{i+1} ou de y_{i+1} exige un registre à décalage à droite de i-bit et un additionneur/soustracteur. Si la fonction $tan^{-1} \; 2^{-i}$ est pré calculé et mémorisé dans un tableau (Tableau 2.1) pour les différentes valeurs de i, un simple additionneur/soustracteur suffit pour calculer z_{i+1} . Chaque itération CORDIC ainsi comporte deux registres à décalage, un tableau de consultation (LUT) et trois additionneurs.

Si la rotation est faite par le même ensemble des angles (avec un signe + ou -), alors le facteur K est une constante, et peut être pré calculer. Par exemple pour tourner par un angle de 30 degré, on suive la séquence des angles qui s'ajoute jusqu'à 30 degré.

$$30 \approx 45.0 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 - 0.2 + 0.1$$

$$= 30.1$$

En réalité, ce qui se produit réellement dans l'algorithme CORDIC est que z est initialisé à 30 degrés et puis, dans chaque opération, le signe du prochain angle de rotation est sélectionné pour essayer de changer le signe de z ; qui est, d_i=sign (z_i), où la fonction de signe est défini pour être -1 ou 1 selon si l'argument est négatif ou non négatif. Sa nous rappelle la division sans remise. Le Tableau 2.2 montre le procédé de sélectionner les signes des angles tournés pour une rotation désirée de +30 degrés. La Figure 2.5 dépeint les étapes premières du procédé de forcer z à zéro.

Tableau 2.2 : les valeurs approximatives de la fonction $\boldsymbol{\alpha_i}$=arctan ($\mathbf{2^{-i}}$), en degré

Pour $0 \leq i \leq 9$.

I	α_i
0	45
1	26.6
2	14
3	7.1
4	3.6
5	1.8
6	0.9
7	0.4

8	0.2
9	0.1

Dans la terminologie CORDIC la règle de sélection précédente pour d_i, qui fait converger z vers zéro, est connue comme **mode de rotation**. Réécriture de l'itération CORDIC, où $\alpha_i = tan^{-1}\ 2^{-i}$:

$$x_{i+1} = x_i \ - \ y_i\ d_i\ \ 2^{-i} \qquad \textbf{(2.30)}$$

$$y_{i+1} = y_i \ + \ x_i\ d_i\ \ 2^{-i} \qquad \textbf{(2.31)}$$

$$z_{i+1} = z_i \ - \ \ d_i\ \ \alpha^i \qquad \textbf{(2.32)}$$

Après m itérations en mode rotation, et quand z (m) tend vers zéro. Nous avons $\sum\alpha_i$ = z, et les équations CORDIC sont :

$$x_m = \text{k (xcos z - ysin z)} \qquad \textbf{(2.33)}$$

$$y_m = \text{k (ycos z + xsin z)} \qquad \textbf{(2.34)}$$

$$z_m = 0 \qquad \textbf{(2.35)}$$

Règle : choisissant $d_i \epsilon \{-1, 1\}$ tel que $z \rightarrow 0$

La constante K dans les équations précédentes est k = 1.646760258121…..ainsi, pour calculer cos z et sin z, on peut commencer par x = 1/K = 0.607252935...et y = 0. Puis, comme z_m tend vers 0 dans les itérations CORDIC en mode rotation, x_m et y_m convergent vers cos z et sin z, respectivement.

Tableau 2.3 : Le choix des signes de la rotation des angles pour forcer z à zéro

i	$z_i - \alpha_i$	z_{i+1}
0	+ 30.0 - 45.0	-15
1	- 15.0 + 26.6	11.6
2	+ 11.6 - 14.0	-2.4
3	- 2.4 + 7.1	4.7
4	+ 4.7 - 3.6	1.1
5	+ 1.1 -1.8	-0.7

6	- 0.7 + 0.9	0.2
7	+ 0.2 - 0.4	-0.2
8	- 0.2 + 0.2	0
9	+ 0.0 - 0.1	-0.1

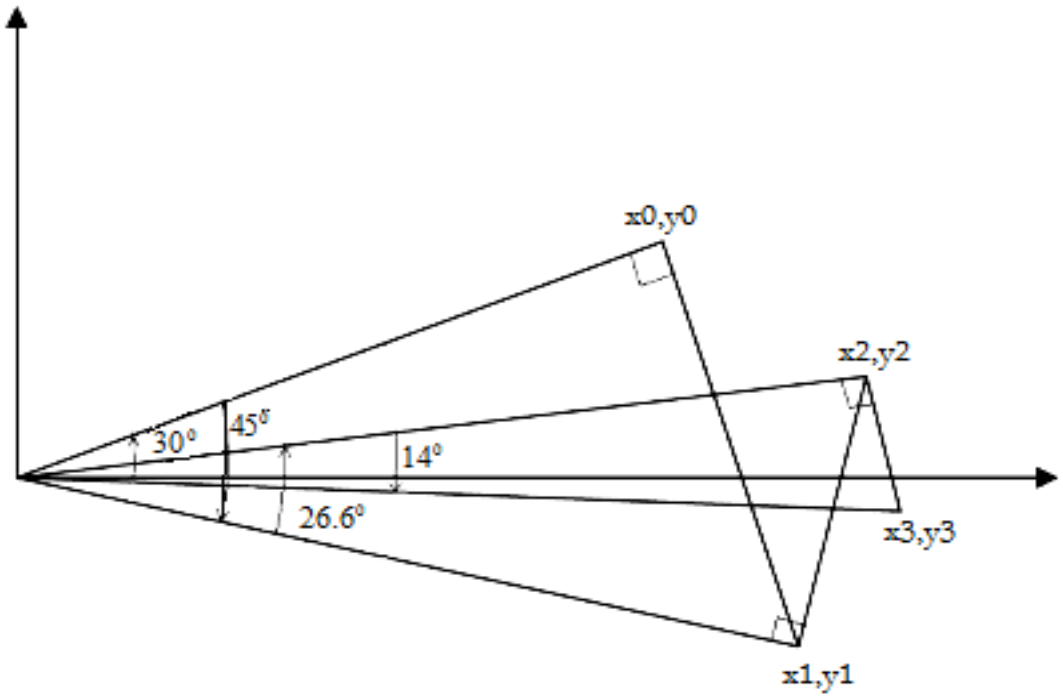

Figure 2.5: les trois premières itérations parmi les 10 principales dans la rotation par 30°,

mode Rotation.

Pour une précision de k bits des fonctions trigonométriques résultantes, on a besoin k itérations CORDIC. La raison est que pour un i grand on peut faire l'approximation $tan^{-1}\ 2^{-i} \approx 2^{-i}$. D'où, pour i > k,

En mode Rotation, la convergence de z à zéro est possible parce que chaque angle dans le Tableau 2.4 est plus que moitié de l'angle précédent ou, d'une manière équivalente, chaque angle est moins que la somme des angles qui le suivent. Le domaine de convergence est -99.7 <z <99.7, où 99.7 est la somme de tout les angles dans le Tableau 2.4. Heureusement, cette gamme comprend les angles de -90 à +90, où [-π/2 à +π/2] en radian. En dehors de la gamme précédente, des identités trigonométriques peuvent être converties en problème, à un qui est dans le domaine de convergence :

$$\cos (z \pm 2j\pi) = \cos z \qquad \textbf{(2.36)}$$

$$\sin(z \pm 2j\pi) = \sin z \quad \textbf{(2.37)}$$

$$\cos(z - \pi) = -\cos z \quad \textbf{(2.38)}$$

$$\sin(z - \pi) = -\sin z \quad \textbf{(2.39)}$$

Notant que ces transformations deviennent particulièrement pratique si les angles sont représentées en multiple de π, de sorte que z = 0.2 réellement signifie que z = 0.2π ou converti en nombres dans le domaine bien facilement.

Dans la deuxième voie par l'utilisation des itérations CORDIC, connue sous le nom de «**mode Vecteur** », le y est rendu plus proche de zéro en choisissant d_i = -sign($x_i y_i$). Après m itérations dans le mode Vecteur $\tan(\sum \alpha_i) = -y/x$.

Ceci signifie que :

$$x_m = k\,[x\cos(\textstyle\sum \alpha_i) - y\sin(\sum \alpha_i)] \quad \textbf{(2.40)}$$

$$x_m = k(x - y\tan(\textstyle\sum \alpha_i))/[1 + tan^2(\sum \alpha_i)]^{1/2} \quad \textbf{(2.41)}$$

$$x_m = k(x + y^2 / x)/ (1 + y^2/x^2) \quad \textbf{(2.42)}$$

$$x_m = k\,(x^2 + y^2)^{1/2} \quad \textbf{(2.43)}$$

Les équations CORDIC sont ainsi :

$$x_m = k\,(x^2 + y^2)^{1/2} \quad \textbf{(2.44)}$$

$$y_m = 0 \quad \textbf{(2.45)}$$

$$z_m = z + \tan^{-1}(y/x) \quad \textbf{(2.46)}$$

Règle : choisissant $d_i \epsilon \{-1, 1\}$ tel que $y \rightarrow 0$

On peut calculer $\tan^{-1} y$ dans le mode Vecteur en commençant par x = 1 et z = 0. Ce calcul converge toujours. [5]

2.4 Etude de la précision de l'algorithme CORDIC

Comme prévu, les algorithmes itératifs calculent les résultats par approximation et la solution contient des erreurs. CORDIC n'est pas une exception et les erreurs sont introduites par une combinaison d'erreurs de quantification et de rapprochement. La précision d'un processeur CORDIC est dépendante de la longueur de mot utilisée pour les trois variables d'entrée x, y et z. ainsi que le nombre d'itérations ou les étapes réalisée.

2.4.1Estimation de la précision CORDIC

Les opérations fondamentales exécutées par un processeur CORDIC est le processus shif et add dont arithmétique à virgule fixe sera introduit d'erreurs.

La précision numérique de l'algorithme CORDIC peut être calculée par l'examinassions des erreurs de troncature et de rapprochement. Erreurs de troncature sont dues à la longueur de mot finie et les erreurs d'approximation sont dues au nombre fini d'itérations. Walther a analysé les itérations x et y indépendamment des itérations z et a conclu que log (n) bits supplémentaires dans les chemins de données peut fournir des n bits de précision. Ce travail a été recalculé par Kota et Cavallaro d'une manière non-indépendant et ont conclu que log (n) +2 bits supplémentaires sont nécessaires pour atteindre bits de précision, après n itérations. [13]

Cette solution représente une limite supérieure de l'erreur dans le processeur CORDIC. Un graphe de cette fonction apparaît dans la figure 3.5 à partir de laquelle on peut voir que, pour atteindre 8 ou 16 bits Précis, les chemins de données internes doivent être les 13 et 22 bits, respectivement.

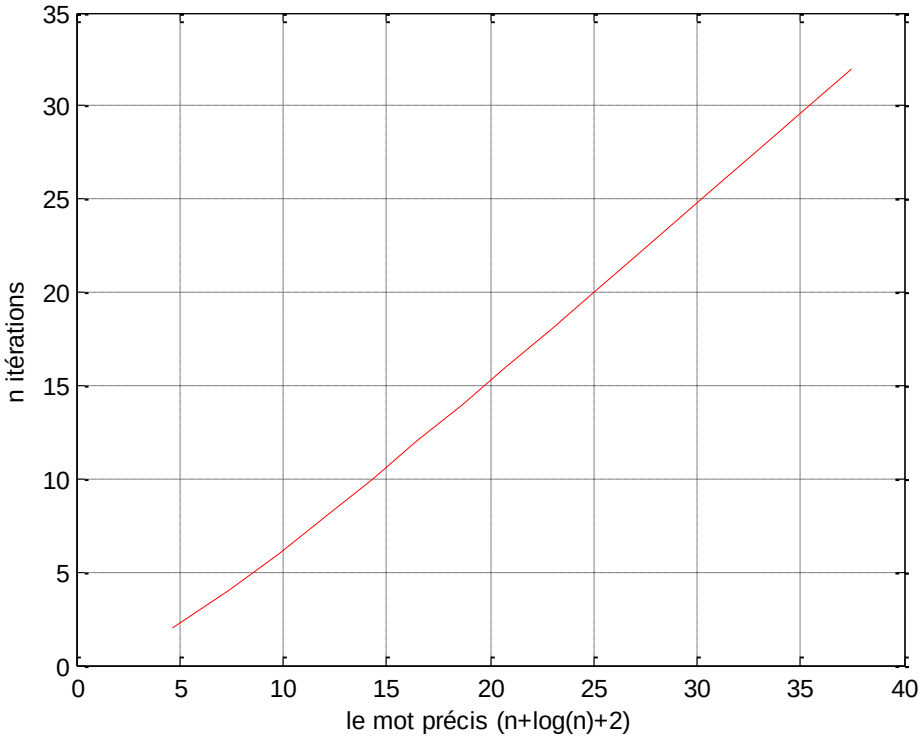

Figure 3.5 La précision numérique du processeur CORDIC

Les résultats des simulations indiquent que n + log n + 2 est une surestimation de chemin de données largeur requise et une réduction de la largeur de chemin de données est possible si le nombre d'itérations est accru.

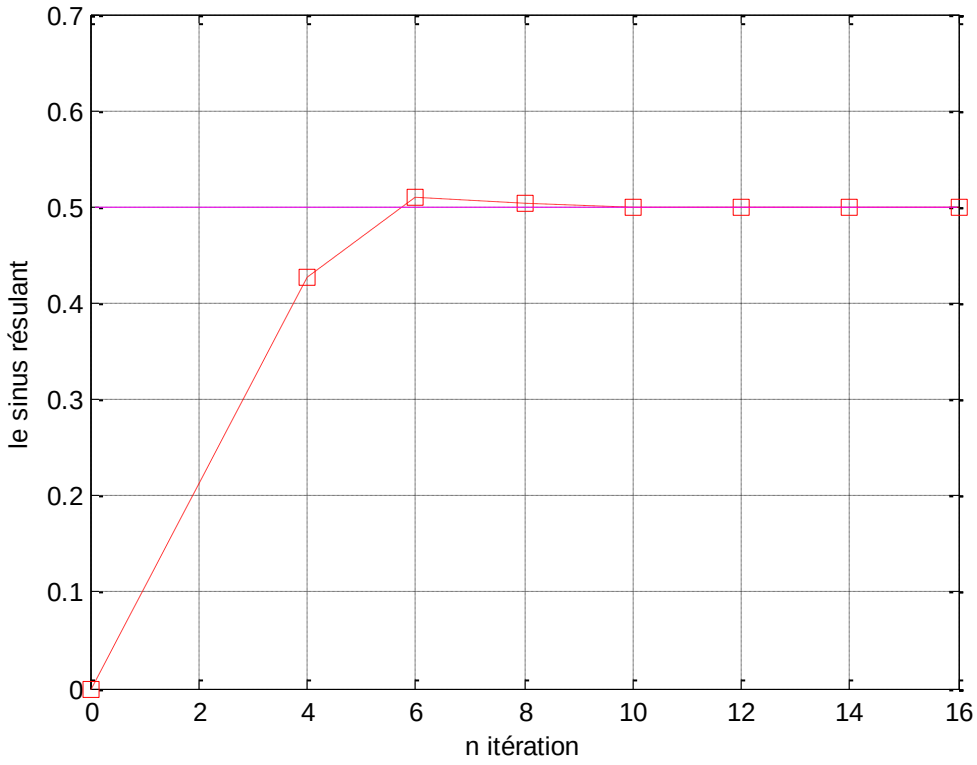

Figure 3.6 comparaison entre la valeur exacte de sinus thêta et les résultats de CORDIC pour différents nombres des itérations (thêta = 30°)

Tableau 2.4 les résultats de simulation de CORDIC pour nombre de bits variables

N bits	**4**	**8**	**12**	**16**	**20**	**24**
Sin(30)	0.5	0.5	0.49951171875	0, 4998779296875	0,499996185302734375	0.499999702345467
Cos(30)	0.875	0,8828125	0. 8671875	0,866119384765625	0.866008758544921875	0.866017896314257

Prenant la valeur exacte de Cos (30°)= 0,866025403784438646767372317075294, on obtient les figure suivants :

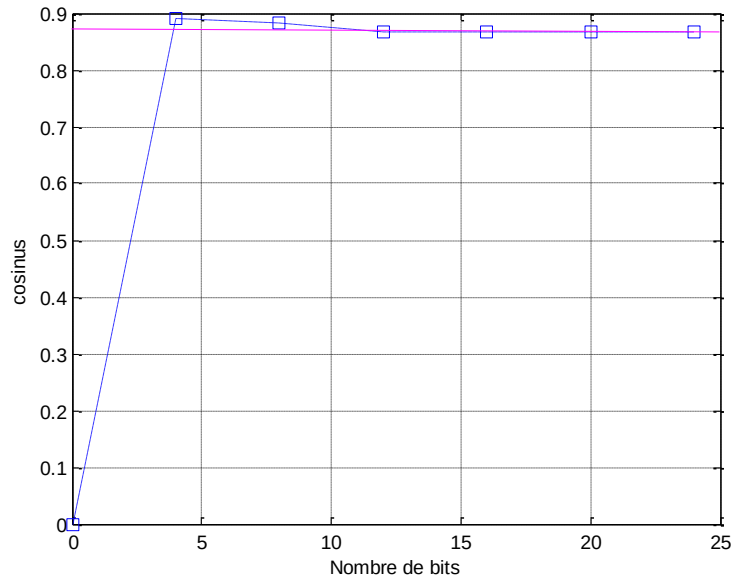

Figure 3.7 comparaison entre la valeur exacte de cosinus thêta et les résultats de CORDIC pour différents nombres de bits (thêta = 30°)

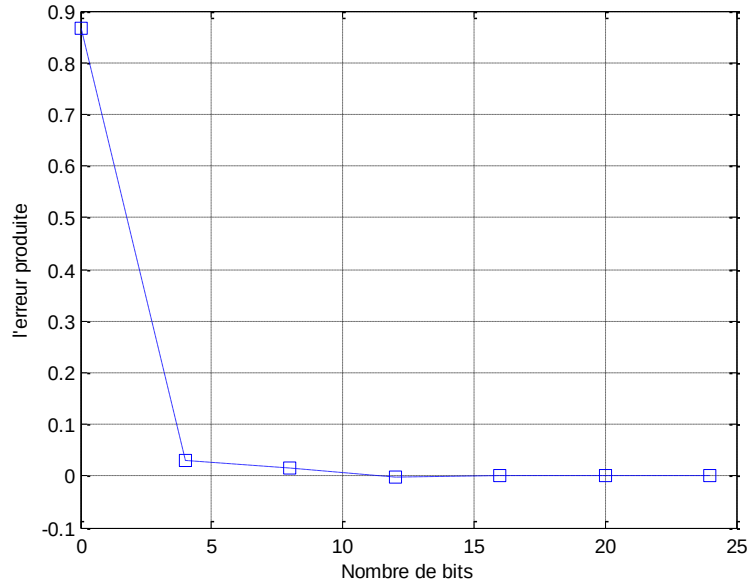

Figure 3.8 L'erreur produit par l'algorithme CORDIC pour nombre de bits variable

D'après les figures 3.7 et 3.8 on remarque que le nombre de bits influx sur le résultat obtenue par le programme VHDL de l'algorithme CORDIC, où l'augmentation de ce dernier traduit par une augmentation de nombre de bits jusqu'à 20bits où la précision atteint le 100%

Chapitre 3 : Architectures de l'Opérateur CORDIC

3.1 Introduction

Les différents choix pour implémenter l'algorithme CORDIC se différencient par les degrés de parallélisme, et les profondeurs de pipeline des additionneurs/soustracteurs utilisés dans l'algorithme. Le choix est guidé par un compromis entre la précision souhaitée, la vitesse de calcul nécessaire et le niveau de complexité demandé par l'application. On peut distinguer trois architectures en fonction du parallélisme sur les opérations et les itérations.
En effet, la mise en parallèle de plusieurs additionneurs permet d'augmenter la vitesse de calcul d'un facteur fixe, mais multiplie d'autant le nombre d'opérateurs élémentaires nécessaires à la réalisation de la fonction.

3.2 CORDIC élémentaire

La figure ci-après explique l'architecture de CORDIC élémentaire. Elle montre les additionneurs/soustracteurs et les registres à décalage. Les additionneurs/soustracteurs effectuent l'addition/soustraction des nombres binaires. Les registres à décalage exécutent l'opération de décalage conformément à l'algorithme. Les constantes correspondant aux valeurs fixes des angles sont obtenues à partir d'une LUT mis en application comme ROM.

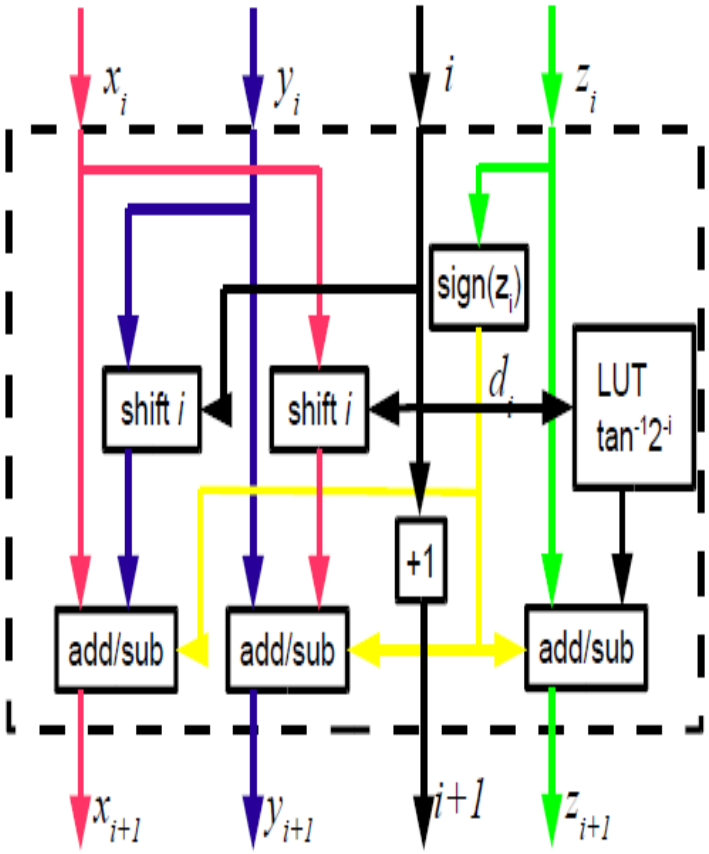

Figure 3 : CORDIC élémentaire

3.3 L'additionneur

Un additionneur est un circuit_logique permettant de réaliser une addition. On dénombre deux types d'additionneurs : parallèle (circuit combinatoire) et série (circuit séquentiel).

3.3.1 Additionneur complet:

La cellule d'additionneur que l'on vient d'étudier ne nécessite que 23 transistors. Elle est donc beaucoup plus optimisée que si elle était réalisée avec des portes classiques. Il existe plusieurs façons de réaliser un additionneur capable d'additionner deux vecteurs de bits à partir d'une telle cellule.

- soit les différents bits du vecteur sont traités séquentiellement (addition série)
- soit ils sont traités en parallèle (addition parallèle)

A part quelques applications particulières, la quasi-totalité des additionneurs modernes travaillent en parallèle.

Entrées			**Sorties**	
A	**B**	**C_0**	**C_1**	**S**
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

On peut remarquer sur la table de vérité que S est le *ou exclusif* des trois entrées A, B et C_0, *i.e.* $S = A \oplus B \oplus C_0$ et que la retenue C_1 vaut 1 dès que deux des trois entrées valent 1, c'est-à-dire $C_1 = AB \vee AC_0 \vee BC_0$.

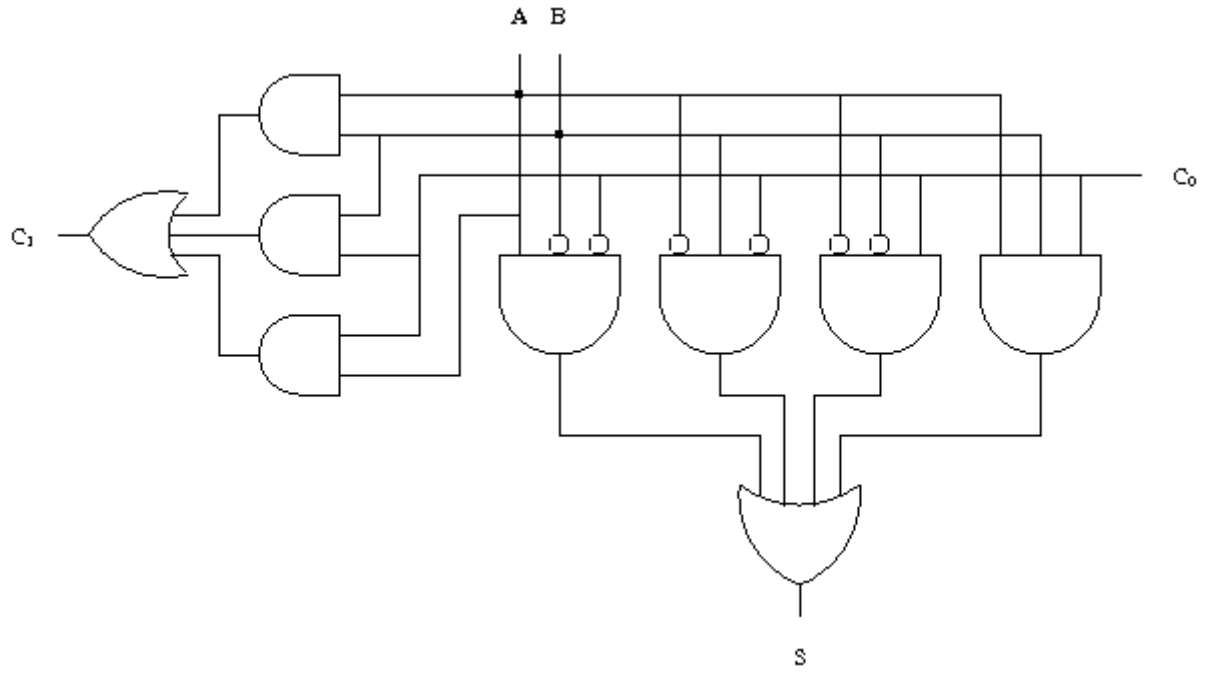

Figure 3.1 Schéma à partir des formules d'un additionneur complet

3.3.2 Additionneur par propagation de retenue

L'additionneur par propagation de retenue est le plus simple. Il est calqué sur l'algorithme manuel pour effectuer l'addition. Les paires de bits sont additionnés colonne par colonne et les retenues sont propagées vers la gauche.

En mettant en cascade k additionneurs complets 1 bits, on construit un additionneur k bits appelé *additionneur par propagation de retenue* car la retenue se propage d'additionneur en additionneur.

L'additionneur est formé de k additionneurs 1 bit numérotés de 0 à k-1 de la droite vers la gauche. L'additionneur i reçoit en entrées les bits A_i et B_i des entrées A et B ainsi que la retenue C_i engendrée par l'additionneur i-1. Il calcule le bit S_i de la somme et la retenue C_{i+1} qui est transmise à l'additionneur i+1.

3.4 Soustracteur

Pour calculer la différence A - B de deux nombre signés A et B, on utilise un circuit qui calcule d'abord l'opposé -B de B puis effectue la somme de A avec -B grâce à un additionneur. Le calcul de -B est réalisé en prenant la négation de B bit à bit puis en ajoutant 1 au résultat obtenu. Ce dernier 1 est en fait ajouté directement à la somme de A et -B en l'injectant comme retenue C_0 à l'additionneur.

Le circuit ci-dessous effectue une somme ou une différence suivant la valeur de la commande Cmd. Si Cmd vaut 0, le circuit calcule la somme A + B. Si, au contraire, Cmd vaut 1, le circuit calcule la différence A - B. En effet, chacune des portes *xor* effectue la négation ou non d'une entrée B_i suivant la valeur de Cmd.

La différence est ici calculée avec un simple additionneur par propagation de retenue mais tout autre additionneur plus sophistiqué aurait pu aussi être utilisé.

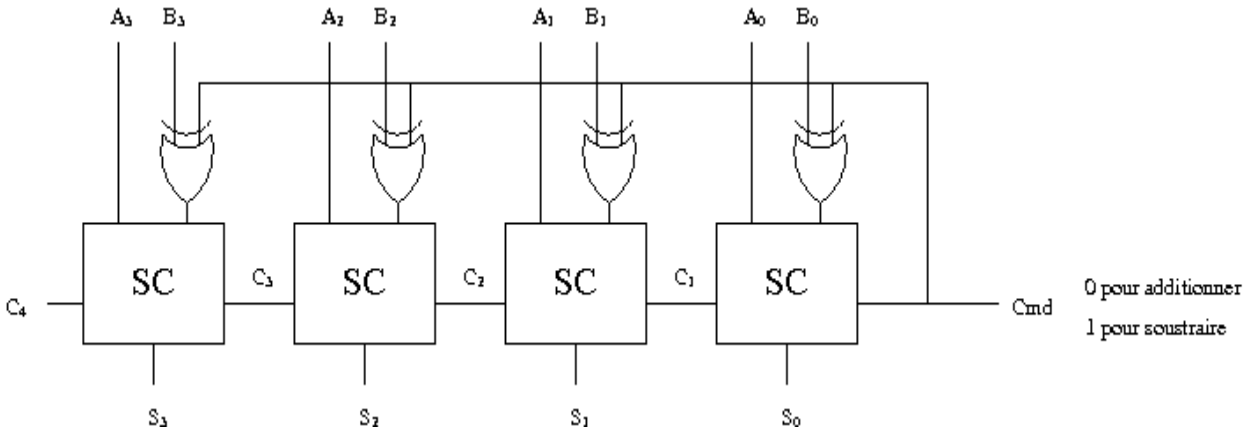

Figure 3.2 Additionneur/Soustracteur 4 bits

3.5 Registre à décalage

Un **registre à décalage** est un registre, c'est-à-dire un ensemble de bascules synchrones, dont les bascules sont reliées une à une, à l'exception de deux bascules qui ne sont pas forcément reliées. À chaque cycle d'horloge, le nombre représenté par ces bascules est mis à jour.

Différents types de registre à décalage existent :

- SISO (*Serial In - Serial Out*) : à chaque cycle d'horloge, on ne peut écrire que dans une bascule (lorsque deux bascules ne sont pas reliées entre elles, c'est celle dont l'entrée n'est pas reliée à une autre bascule), et on ne peut lire qu'une autre (lorsque deux bascules ne sont pas reliées entre elles, c'est celle dont la sortie n'est pas reliée à une autre bascule)
- SIPO (*Serial In - Parallel Out*) : à chaque cycle d'horloge, on ne peut écrire que dans une bascule (lorsque deux bascules ne sont pas reliées entre elles, c'est celle dont l'entrée n'est pas reliée à une autre bascule), mais on peut lire les valeurs de toutes les bascules
- PISO (*Parallel In - Serial Out*) : à chaque cycle d'horloge, on peut forcer la valeur de toutes les bascules, mais on ne peut en lire qu'une (lorsque deux bascules ne sont pas reliées entre elles, c'est celle dont la sortie n'est pas reliée à une autre bascule)
- PIPO (*Parallel In - Parallel Out*) : à chaque cycle d'horloge, on peut forcer la valeur de toutes les bascules, et on peut toutes les lire

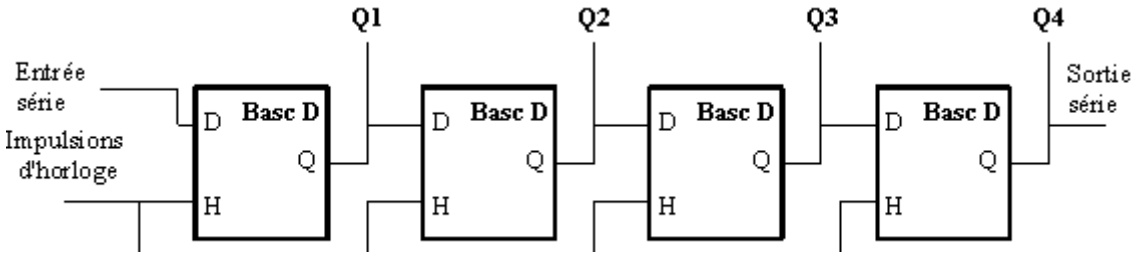

Figure 3.3 Registre a décalage SISO ou SIPO

Le registre à décalage réversible est un registre à décalage où le décalage s'effectue vers la droite ou vers la gauche en fonction du niveau logique appliqué à l'entrée « Sens de décalage ».

L'utilisation des registres a décalage :

- division par une puissance de deux (décalage vers la gauche ou la droite)
- tampons pour la réception de données (FIFO : *first in – first out*)
- compteur, temporisateur

3.6 Architecture série de l'algorithme CORDIC

On met ici en œuvre des opérateurs arithmétiques parallèles qui réalisent simultanément le calcul des trois équations de l'algorithme, le calcul des n itérations étant enchaîné séquentiellement. La figure 3.4 illustre ce type d'architecture.
Avant de commencer le calcul il faut réaliser la phase d'initialisation qui consiste à charger les registres X, Y et Z avec leur valeur respective x_0, $y_0$et z_0.
Les opérateurs shiften sont des décaleurs programmables dont les valeurs de décalage à appliquer aux variables X et Y à chaque itération dépendent de la base de décomposition. L'élément de base α_i est sélectionné en accord avec le rang de l'itération en cours de traitement et ajouté ou retranché à la variable courante z_i.
L'organe de contrôle FSM (finite state machine) élabore les signaux de commande des additionneur-soustracteurs (choix de l'opération), des décaleurs (valeur du décalage), en fonction des signes des variables courantes y_i, et z_i.
Chaque itération est calculée en deux temps. Les données$n$ x_i, y_i, z_i, sont d'abord transférées des registres X, Y, Z vers l'unité de calcul par l'intermédiaire des décaleurs. Le deuxième temps comprend le calcul puis le stockage des variables résultats x_{i+1}, y_{i+1}, z_{i+1}, dans les registres X, Y et Z. C'est cette dernière étape de calcul qui constitue la chaîne critique de l'architecture et qui fixe ses performances de vitesse. La fréquence de fonctionnement est limitée par le temps nécessaire à l'opérateur arithmétique pour réaliser une opération d'addition ou de soustraction.
La période de calcul correspond à n fois le temps caractéristique de la chaîne critique, où n est le nombre d'itérations.
Il faut noter qu'il est nécessaire de stocker les valeurs des arguments angulaires utilisés dans le calcul sur la variable auxiliaire z dans une mémoire dans notre cas c'est atan32.

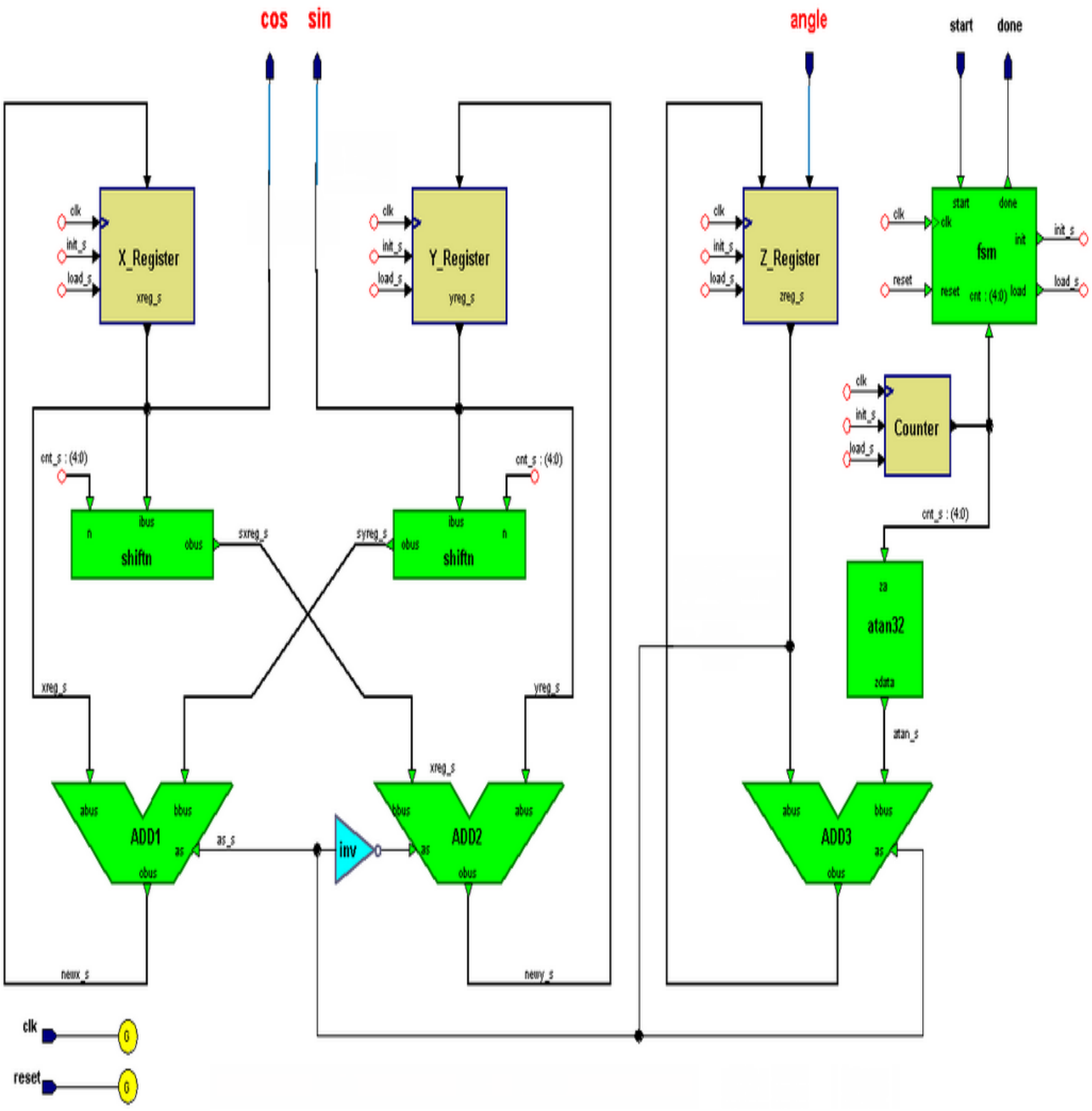

Figure 3.4 Architecture série de l'algorithme CORDIC

3.6.1Machine d'état

L'architecture série de CORDIC est basée sur une machine d'état au niveau de l'FSM + le Counter ; le table de vérité correspond a cette machine est comme suit :

	reset	Start	cont	init	load	done
S0	0	0	0	1	0	0
S1	0	1	0	0	1	0
S2	0	0	1	0	0	1

Ou S0, S1 et S2 sont les états de la machine ;

Reset; start ; cont ; init ; load ; done : les signaux d'entrée/sortie de la machine ;la figure montre l'emplacement de ces derniers ,

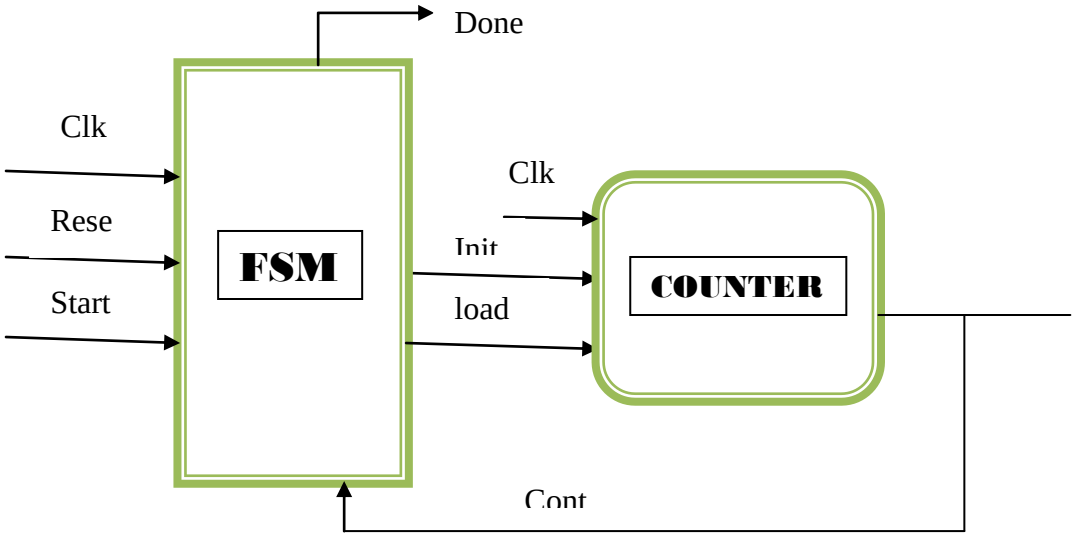

Figure 3.5 : Représentation des signaux d'entée/sortie de la machine d'état

La machine d'état est montrée sur la figure 3.6 :

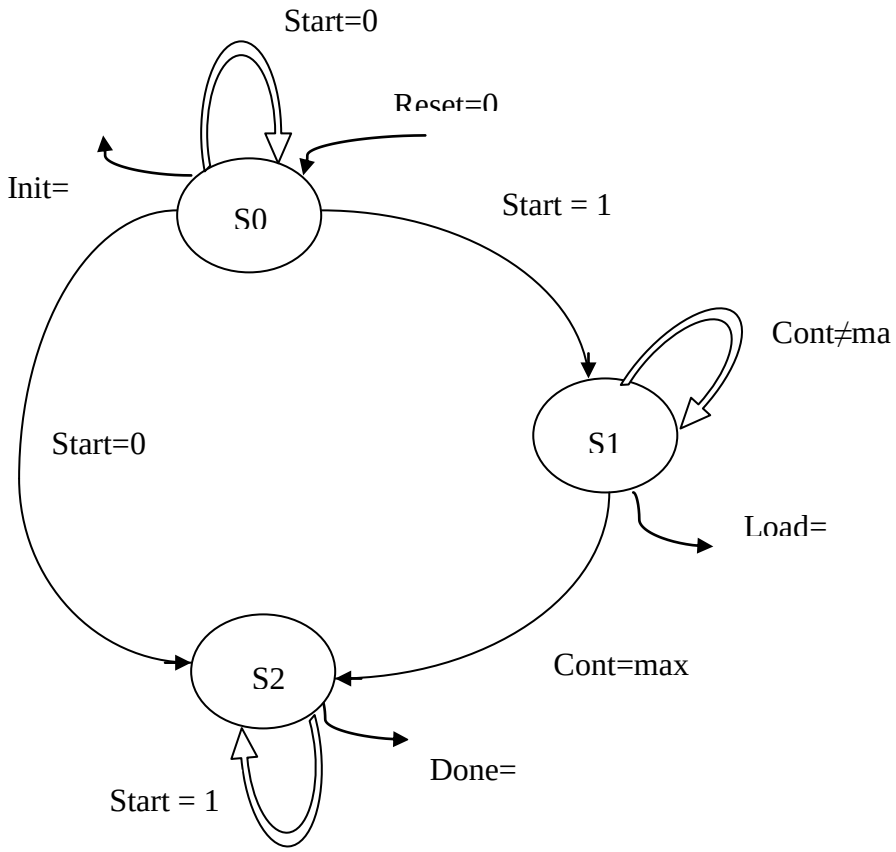

Figure 3.6 : machine d'état de l'architecture série de l'algorithme CORDIC

Avantages :

- la structure matérielle n'est pas complexe (pas d'encombrement).
- Consomme peu de puissance.

Inconvénients :

- Un nombre maximum de cycles d'horloge sont nécessaires pour calculer la sortie, donc le temps de calcul est important.

3.7 Architecture parallèle

Au lieu de mettre en mémoire tampon la sortie d'une itération et d'employer les mêmes ressources de nouveau, on pourrait simplement monter en cascade le CORDIC itératif, qui signifie reconstruire la structure CORDIC fondamentale pour chaque itération. En conséquence, la sortie d'une étape est l'entrée de la prochaine, suivant les indications du la figure 3.5, et face aux étapes séparées deux simplifications deviennent possibles. D'abord, les fonctionnements des décaleurs pour chaque opération peuvent être exécutés en câblant les connexions entre les étapes convenablement. En second lieu, il n'y a aucun besoin de changer les constantes et ceux peuvent pour cette raison être aussi bien câblés. Le modèle parallèle se compose seulement des composants combinatoires et calcule une valeur de sinus selon le cycle d'horloge.

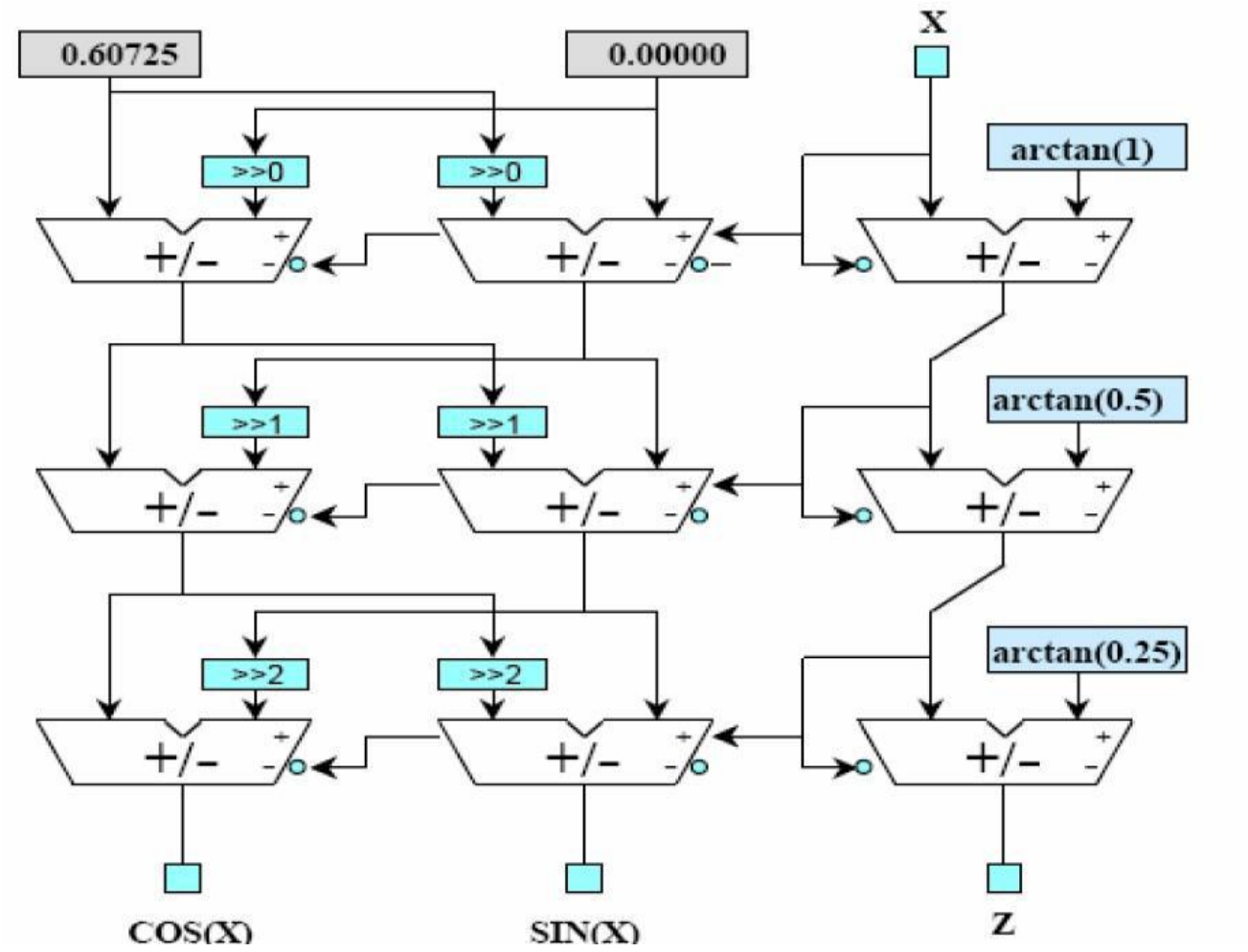

Figure 3.5 L'architecture parallèle de l'algorithme CORDIC

Avantages **:**

- Plus rapide que la structure itérative

- Les décaleurs ont des tailles fixes donc ils peuvent être mis en application dans le câblage.
- Des constantes peuvent être câblées au lieu d'exiger l'espace de stockage.

Inconvénients :

- Occupation de surface plus grande plusieurs Cordics élémentaires cablés (plus d'encombrement)
- La consommation d'énergie est plus élevée par rapport l'architecture précédente

3.8 Architecture pipeline de CORDIC :

Le même principe avec l'architecture parallèle de tel sorte quand 'on utilise plusieurs cellule CORDIC en cascade mais dans cette architecture entre chaque étage on insère des registres de piplinage qui permettent d'isoler la partie calcul de la partie mémorisation des variables intermédiaires.

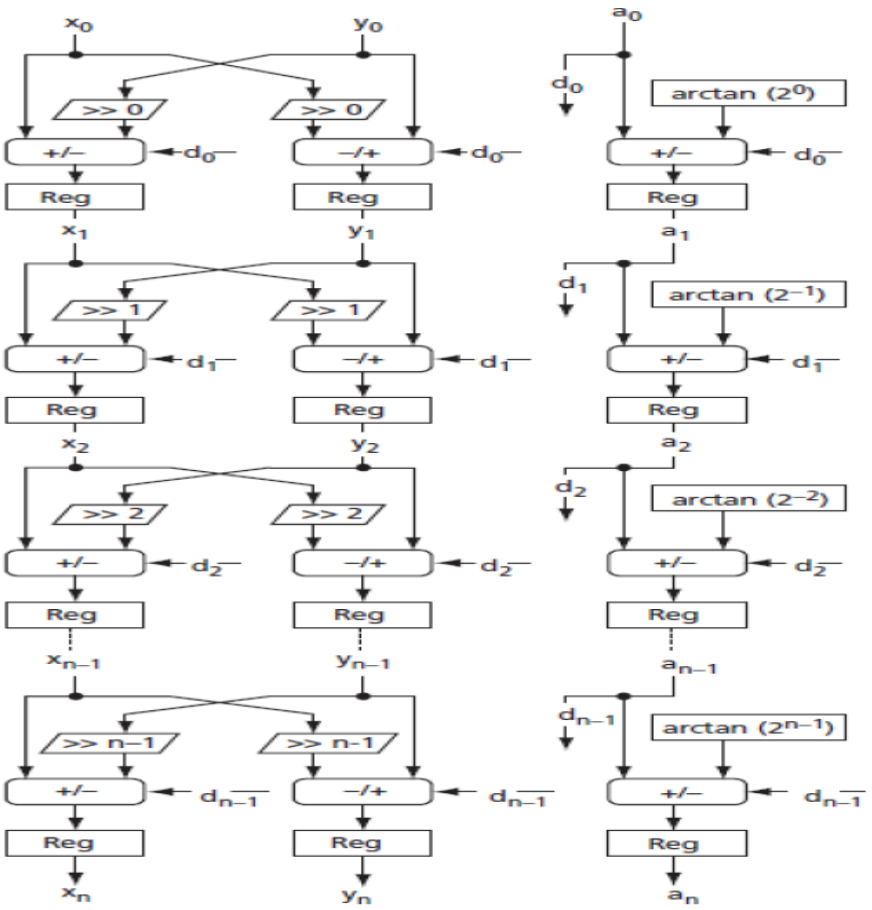

Figure 3.4 L'architecture pipline de l'algorithme CORDIC

Avantages

- Présente un retard considérable, mais le temps de traitement est plus réduit que dans le processus itératif.

Inconvénients

- La complexité matérielle la surface occupée est plus grand que dans l'architecture parallèle.
- Consomme plus de puissance que les deux précédentes architectures

Chapitre 4 : Circuits FPGA

4.1 Introduction

Les FPGA (Field Programmable Gate Arrays ou "réseaux logiques programmables") sont des composants entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. L'avantage de ce genre de circuit est sa grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court.

Le progrès de ces technologies permet de faire des composants toujours plus rapides et à plus haute intégration, ce qui permet de programmer des applications importantes. Cette technologie permet d'implanter un grand nombre d'applications et offre une solution d'implantation matérielle à faible coût pour des compagnies de taille modeste , le coût de développement d'un circuit intégré spécifique implique un trop lourd investissement.

Les utilisations sont nombreuses, on en cite

- Prototypage de nouveaux circuits (nouveaux microprocesseurs pour ordinateur, par exemple),

- Fabrication de composants spéciaux en petite série.

- Dans les systèmes embarqués de commande temps réel : l'avionique ; Chaque carte de commande de vol d'un Airbus emploie plusieurs FPGA 'Actel'. Les routeurs de réseau informatique emploient plusieurs FPGA ; les systèmes de l'informatique industrielle aussi.

- Adaptation aux besoins rencontrés lors de l'utilisation (par exemple dans des satellites ou des sondes spatiales).

- d'une manière plus prospective, on se rapproche de la vie artificielle avec une intelligence capable d'apprentissage et d'adaptabilité.

On trouve déjà des FPGA dans des éléments de transmission haut débit (Alcatel, Cisco...).

4.2 Architecture des FPGA

4.2.1 Description des composants FPGA

Les FPGA (Field Programmable Gate Arrays ou "réseaux logiques programmables") sont des composants VLSI (Very Large Scale Integration). Ils sont programmables par l'utilisateur et essentiellement constitués de trois parties :

– une matrice de blocs logiques configurables CLB (Configurable Logic Bloc) ;

– des blocs d'entrée/sortie configurables IOB ;

– un réseau d'interconnexions programmables.

La figure 4.1 présente l'architecture générique d'un FPGA.

Il y a plusieurs constructeurs de composants FPGA tels que Actel, Xilinx et Altera. Ces constructeurs utilisent différentes technologies pour la réalisation des FPGA. Parmi ces technologies, celles qui assurent une reprogrammation des FPGA sont les plus intéressantes étant donné qu'elles permettent une grande flexibilité de conception. Les technologies reprogrammables les plus utilisées pour les FPGA sont les suivantes :

– la technologie Static RAM (SRAM – Static Random Access Memory), pour laquelle, les connexions sont réalisées en rendant les transistors passants, ce qui permet une reconfiguration rapide du circuit FPGA. Cependant, la surface nécessaire pour la SRAM est un inconvénient ;

– la technologie Flash qui est limitée en nombre de reconfigurations et possède un temps de reconfiguration plus long que celui de la technologie SRAM. Cependant, cette technologie garde sa configuration même si l'alimentation est enlevée. Par conséquent, un FPGA à base de technologie Flash déjà programmé est prêt à fonctionner dès sa mise sous tension[15].

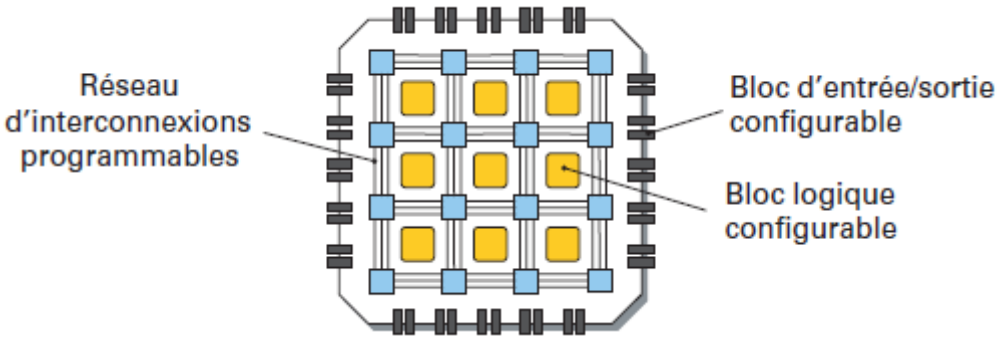

Figure.4.1 : Architecture interne d'un FPGA.

Les circuits FPGA du fabricant Xilinx utilisent deux types de cellules de base :

- Les cellules d'entrées/sorties appelés IOB (input output bloc),
- Les cellules logiques appelées CLB (configurable logic bloc). Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurable.

4.2.2 Les CLB (configurable logic bloc)

Les blocs logiques configurables sont les éléments déterminants des performances du FPGA. Chaque bloc est composé d'un bloc de logique combinatoire composé de deux générateurs de fonctions à quatre entrées et d'un bloc de mémorisation synchronisation composé de deux bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La figure ci-dessous, nous montre le schéma d'un CLB.

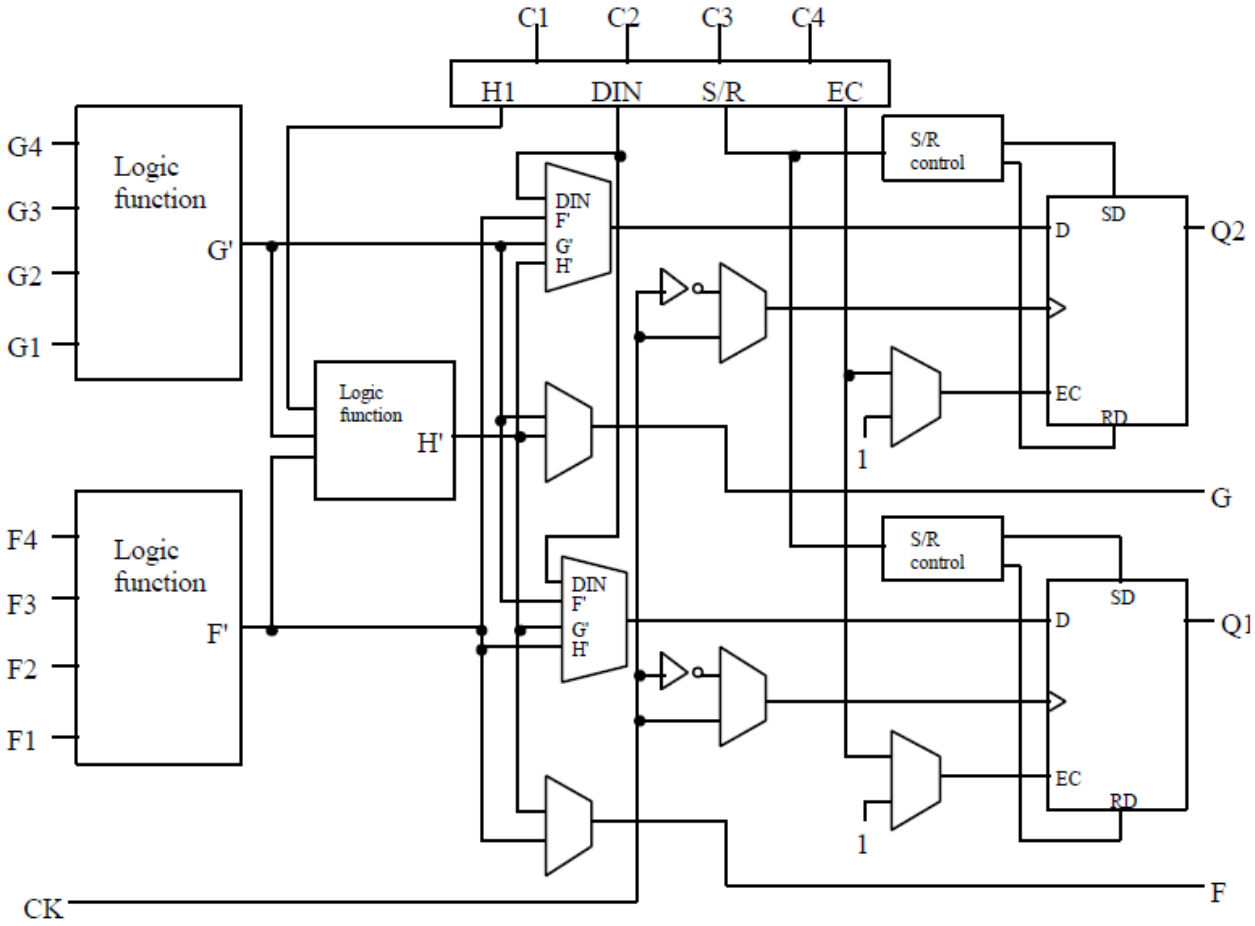

Figure4.2 : Cellules logiques (CLB)

D'abord le bloc logique combinatoire qui possède deux générateurs de fonctions F' et G' à quatre entrées indépendantes (F1...F4, G1...G4), lesquelles offrent aux concepteurs une flexibilité de développement importante car la majorité des fonctions aléatoires à concevoir n'excède pas quatre variables. Les deux fonctions sont générées à partir d'une table de vérité câblée inscrite dans une zone mémoire, rendant ainsi les délais de

propagation pour chaque générateur de fonction indépendants de celle à réaliser. Une troisième fonction H' est réalisée à partir des sorties F' et G' et d'une troisième variable d'entrée H1 sortant d'un bloc composé de quatre signaux de contrôle H1, Din, S/R, Ec. Les signaux des générateurs de fonction peuvent sortir du CLB, soit par la sortie X, pour les fonctions F' et G', soit Y pour les fonctions G' et H'. Ainsi un CLB peut être utilisé pour réaliser

- Deux fonctions indépendantes à quatre entrées indépendantes
- Ou une seule fonction à cinq variables
- Ou deux fonctions, une à quatre variables et une autre à cinq variables.

L'intégration de fonctions à nombreuses variables diminue le nombre de CLB nécessaires, les délais de propagation des signaux et par conséquent augmente la densité et la vitesse du circuit. Les sorties de ces blocs logiques peuvent être appliquées à des bascules au nombre de deux ou directement à la sortie du CLB (sorties X et Y). Chaque bascule présente deux modes de fonctionnement : un mode 'flip-flop' avec comme donnée à mémoriser, soit l'une des fonctions F', G', H' soit l'entrée directe DIN. La donnée peut être mémorisée sur un front montant ou descendant de l'horloge (CLK). Les sorties de ces deux bascules correspondent aux sorties du CLB XQ et YQ. Un mode dit de " verrouillage " exploite une entrée S/R qui peut être programmée soit en mode SET, mise à 1 de la bascule, soit en Reset, mise à zéro de la bascule. Ces deux entrées coexistent avec une autre entrée appelée le global Set/Reset. Cette entrée initialise le circuit FPGA à chaque mise sous tension, à chaque configuration, en commandant toutes les bascules au même instant soit à '1', soit à '0'. Elle agit également lors d'un niveau actif sur le fil RESET lequel peut être connecté à n'importe quelle entrée du circuit FPGA [15].

Un mode optionnel des CLB est la configuration en mémoire RAM de 16x2bits ou 32x1bit. Les entrées F1 à F4 et G1 à G4 deviennent des lignes d'adresses sélectionnant une cellule mémoire particulière. La fonctionnalité des signaux de contrôle est modifiée dans cette configuration, les lignes H1, DIN et S/R deviennent respectivement les deux données D0, D1 (RAM 16x2bits) d'entrée et le signal de validation d'écriture WE. Le contenu de la cellule mémoire (D0 et D1) est accessible aux sorties des générateurs de fonctions F' et G'. Ces données peuvent sortir du CLB à travers ses sorties X et Y ou alors en passant par les deux bascules.

4.2.3 Les IOB (input output bloc)

Les blocs entrée/sortie permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance).

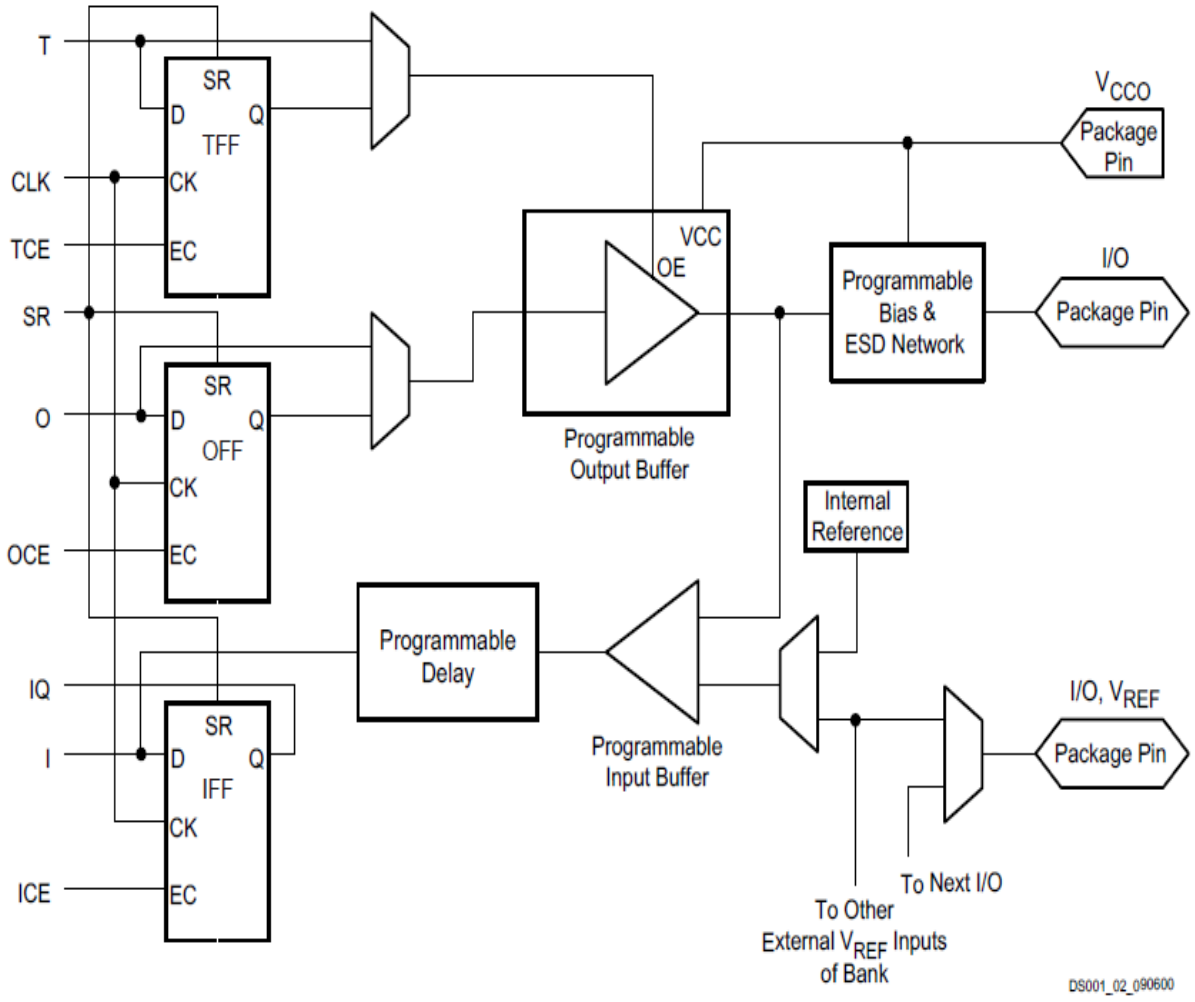

Figure 4.3: Input Output Block (IOB)

Les Ports d'entrée/sortie totalement programmables sont généralement composés de :

• Seuil d'entrée TTL ou CMOS

• Slew-rate programmable

• Buffer de sortie programmable en haute impédance

• Entrées et sorties directes ou mémorisées

• Inverseur programmable

4.2.3.1 La configuration en entrée

Le signal d'entrée traverse un buffer qui selon sa programmation peut détecter soit des seuils TTL ou soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée. Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier [15].

4.2.3.2 La configuration en sortie

Nous distinguons les possibilités suivantes :

- inversion ou non du signal avant son application à l'IOB.

- synchronisation du signal sur des fronts montants ou descendants d'horloge.

- mise en place d'un " pull-up " ou " pull-down " dans le but de limiter la consommation des entrées/sorties inutilisées,

- signaux en logique trois états ou deux états. Le contrôle de mise en haute impédance et la réalisation des lignes bidirectionnelles sont commandés par le signal de commande Out Enable lequel peut être inversé ou non. Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

4.2.4 Les interconnexions

Les connexions internes dans les circuits FPGA sont composées de segments métallisés.

Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexions selon la longueur et la destination des liaisons [15].

4.2.4.1 Les interconnexions à usage général

Ce système fonctionne en une grille de cinq segments métalliques verticaux et quatre segments horizontaux positionnés entre les rangées et les colonnes de CLB et de l'IOB.

Des aiguilleurs appelés aussi matrices de commutation sont situés à chaque intersection. Leur rôle est de raccorder les segments entre eux selon diverses configurations, ils assurent ainsi la communication des signaux d'une voie sur l'autre. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre. Pour éviter que les signaux traversant les grandes lignes ne soient affaiblis, nous trouvons généralement des buffers implantés en haut et à droite de chaque matrice de commutation.

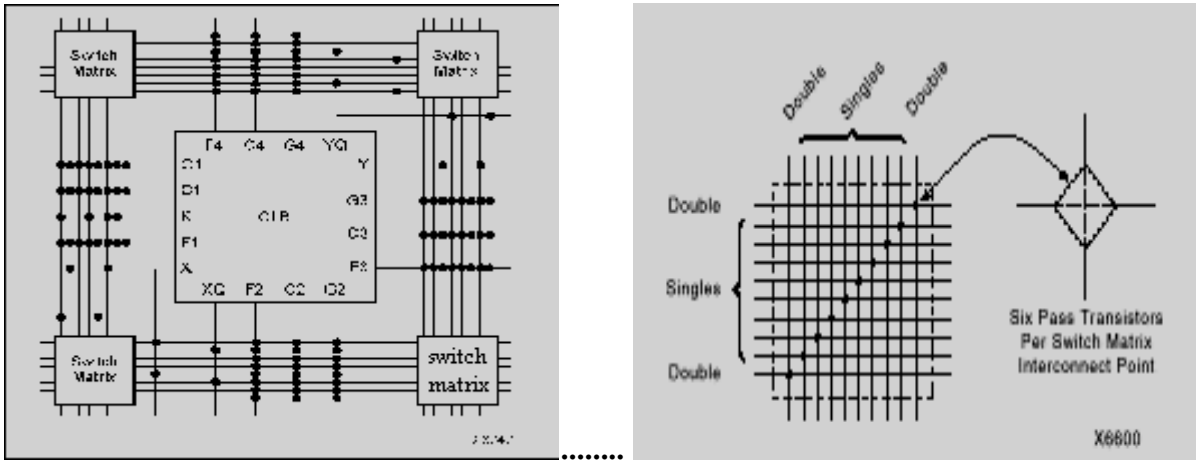

Figure4.4 : Connexions à usage général et matrice de commutation

4.2.4.2 Les interconnexions directes

Ces interconnexions permettent l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en termes de vitesse et d'occupation du circuit. De plus, il est possible de connecter directement certaines entrées d'un CLB aux sorties d'un autre.

Pour chaque bloc logique configurable, la sortie X peut être connectée directement aux entrées C ou D du CLB situé au-dessus et les entrées A ou B du CLB situé au-dessous. Quant à la sortie Y, elle peut être connectée à l'entrée B du CLB placé immédiatement à sa droite. Pour chaque bloc logique adjacent à un bloc entrée/sortie, les connexions sont possibles avec les entrées I ou les sorties O suivant leur position sur le circuit.

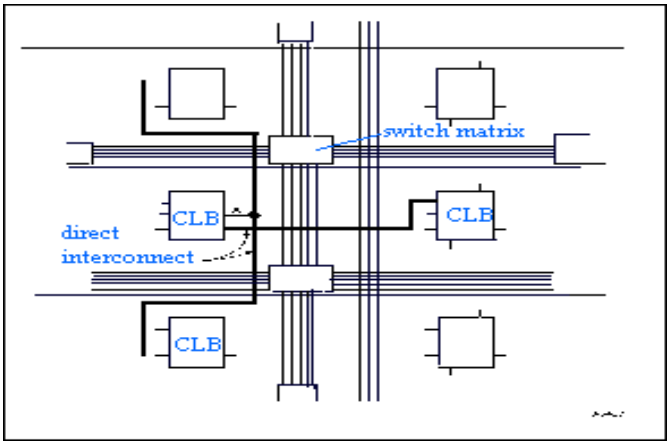

Figure4.5 : Les interconnexions directes

4.2.4.3 Les longues lignes

Les longues lignes sont de longs segments métallisés parcourant toute la longueur et la largeur du composant, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues lignes permettent d'éviter la multiplicité des points d'interconnexion.

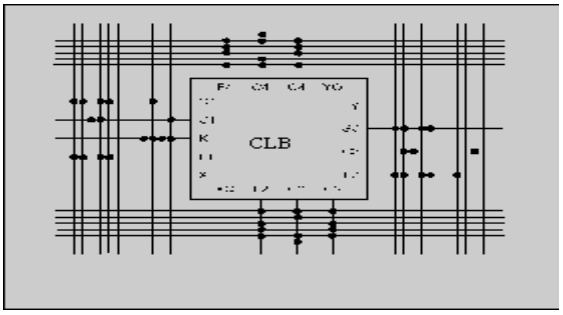

Figure4.6: Les longues lignes

4.4 Flot de conception

Face à l'augmentation de la complexité des FPGA actuels, dotés pour les modèles haut de gamme de plusieurs millions de portes et intégrant un ou plusieurs cœurs de processeurs, les fabricants comme Altera, Xilinx, Actel ou Lattice ont, petit à petit, investi dans leur propre flot de conception. Au point qu'aujourd'hui, le marché se trouve dans une position paradoxale. En effet, le nombre de projets à base d'Asic est en chute régulière (de 10500 conceptions en 1996 à moins de 2500 prévues en 2005, selon une étude récente de la société d'analyse de marché American Technology Research) alors que, parallèlement, le nombre de

projets à base de FPGA est en forte augmentation: 300000 projets en 2001, 500000 en 2004, selon Dataquest. Or, d'un côté, les acteurs traditionnels de la CAO ont bâti toute leur croissance et leur modèle économique sur le marché de l'Asic; de l'autre, les fabricants de FPGA ont construit leur succès sur la vente de composants, tout en investissant au fur et à mesure de la complexité croissante de leur puce dans des outils de développement performants. Le déséquilibre de la situation réside dans le fait que ces fournisseurs de puces programmables ont toujours pratiqué la fourniture gratuite ou quasi gratuite de leurs outils, alors qu'aujourd'hui ces derniers atteignent des niveaux de sophistication très élevés. Par exemple, chez Altera le prix de licence de l'environnement Quartus II est de 2 000 $, chez Xilinx ISE 6i démarre à 695$ et chez Actel Libero est proposé à un prix de licence par siège de 995$, sans compter les versions allégées qui sont téléchargeables gratuitement via Internet.

Tableau4.1 : Flots de conception des fabricants de FPGA

Flots de conception des fabricants de FPGA

Editeur	Actel	Altera	Lattice	Xilinx
Nom du flot	**Libero 5.0**	**Quartus II 3.0**	**IspLever 3.1**	**ISE 6.1i**
Points marquants des nouvelles versions	Vue graphique de la conception au niveau physique et logique; éditeur de paramétrage des E/S; outil de calcul de la consommation du circuit (SmartPower)	Compilation incrémentale; conception et implantation indépendantes de modules (fonction logic clock); outil graphique de connexion d'IP et de cœurs de processeurs (SOPC Builder)	Outil de gestion de projet; simulateur propre avec édition et visualisation des chronogrammes; outil d'analyse et d'estimation de temps critiques sans recompilation	Préplacement de blocs intégrant des contraintes de timing; assignation automatique des broches du FPGA pour le circuit imprimé; outil de débogage temps réel avec déclenchement croisé avec le logiciel
Principaux outils tiers utilisés dans le flot	**Simulation** ModelSim (Mentor) **Génération de testbench** Wave Former Lite 9.0 (SynpaticCAD) **Synthèse** Leonardo, Precision (Mentor), Synplify 7.3 (Edition Actel de Synplify de Synplicity) **Synthèse physique** Palace (Magma)	**Simulation** ModelSim (Mentor), NC Sim (Cadence), Sirocco (Synopsys) **Synthèse** Design Compiler (Synopsys), Leonardo, Precision (Mentor), Synplify FPGA (Synplicity) **Analyse statique de délais** Prime Time (Synopsys)	**Simulation** ModelSim (Mentor) **Synthèse** Synplify FPGA (Synplicity)	**Simulation** NC Sim, Verilog XL (Cadence), ModelSim (Mentor), VCS, Sirocco (Synopsys) **Synthèse** Synplify (Synplicity), Leonardo, Precision (Mentor), FPGA Compiler, Design Compiler (Synopsys) **Synthèse physique** Palace (Magma)

4.4.1 Vue d'ensemble du flot de conception

Un flot de conception est une séquence d'opérations permettant d'obtenir un circuit concret, implémenté dans une technologie donnée, à partir de sa description. Un flot de conception typique de circuit numérique est illustré à la figure 4.9. Ce flot est détaillé dans les sections suivantes.

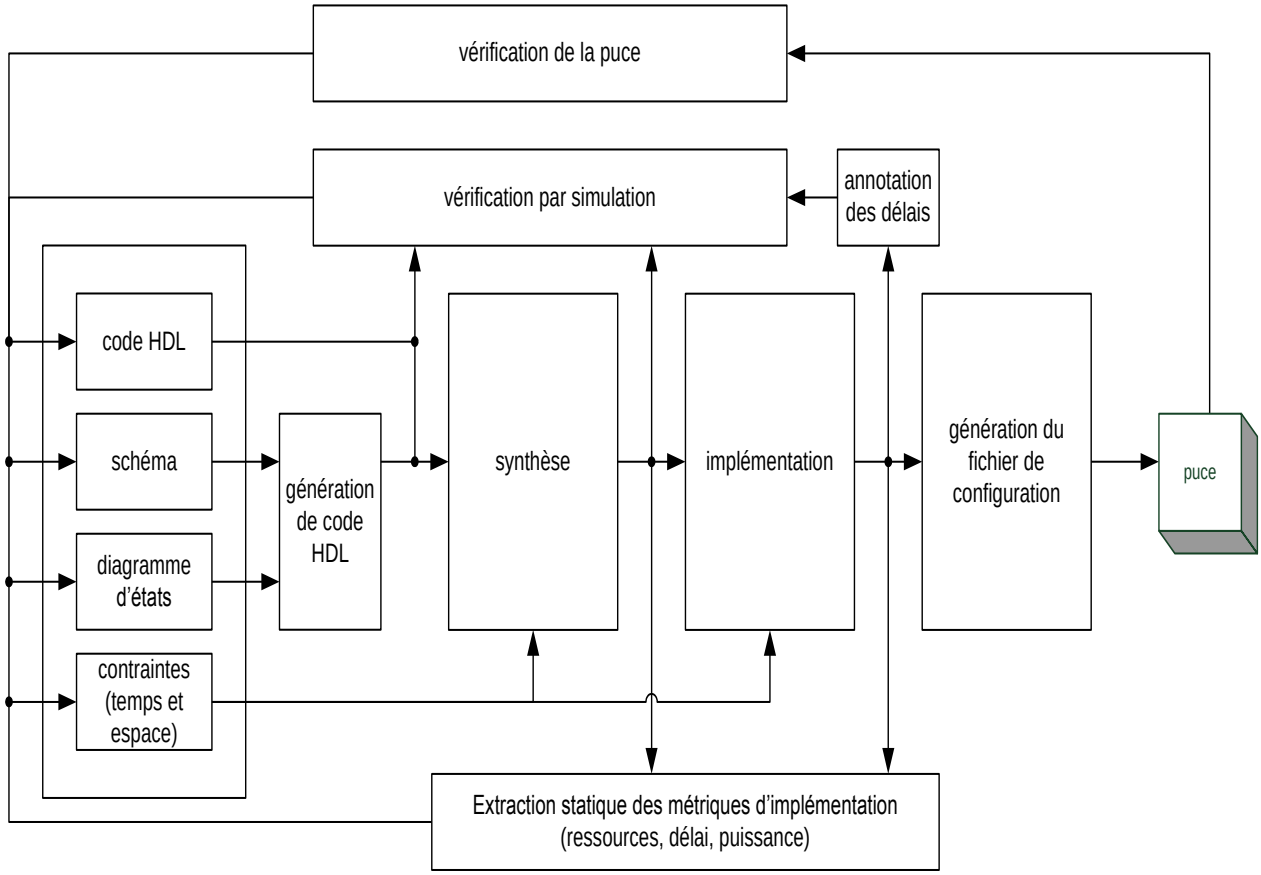

Figure 0-9 - flot de conception d'un circuit numérique

4.4.1.1 Description : code HDL :

La description du module peut se faire par une combinaison de code HDL dans plusieurs langages, de schémas, diagrammes d'états ou encore flots de données. Dans le premier cas, on peut utiliser un simple éditeur de texte pour écrire la description en HDL. Dans les deux autres, un programme de traduction génère une description en HDL à partir du schéma ou du diagramme entré dans l'outil.

4.4.1.2 Simulation fonctionnelle d'un modèle VHDL

La simulation du code HDL permet au concepteur de vérifier que la description est conforme aux spécifications. Cette étape est très importante et requiert une grande proportion de l'effort de design. Un compilateur lit le code HDL et vérifie que la syntaxe du langage est respectée. Il génère une description intermédiaire du code qui peut ensuite être exécutée par un simulateur.

Les simulateurs affichent en général les résultats de la simulation sous la forme d'un chronogramme. Il est possible en général d'observer les entrées et les sorties, ainsi que des signaux internes du circuit qui est simulé. Certains simulateurs permettent aussi de voir les signaux sous la forme d'un tableau, un peu à la manière d'une table de vérité. Les simulateurs peuvent aussi afficher des messages d'avertissement et d'erreur à la console.

4.4.1.3 La synthèse :

La synthèse d'un circuit consiste à traduire la description du circuit en blocs disponibles dans la technologie utilisée. Par exemple, pour un circuit décrit avec un schéma et qui doit être réalisé sur un FPGA, le processus de synthèse convertit et regroupe les portes logiques du schéma en composantes réalisables sur le FPGA choisi.

4.4.1.4 L'implémentation

L'implémentation du circuit est divisée en quatre sous étapes:

• la transformation (mapping) : regrouper les composantes obtenues lors de la synthèse dans des blocs spécifiques du FPGA;

• la disposition (placement) : choisir des endroits spécifiques sur le FPGA où disposer les blocs utilisés, et choisir les pattes du FPGA correspondant aux ports d'entrée et de sortie;

• le routage (routing) : établir des connexions électriques entre les blocs utilisés; et,

• la configuration (configuration) : convertir toute cette information en un fichier pouvant être téléchargé sur le FPGA pour le programmer

Chapitre 5: Implémentations sur FPGA de l'opérateur CORDIC

5.1 Introduction

Le logiciel Xilinx ISE est un outil de conception de circuit pour FPGA de Xilinx. Ce logiciel permet essentiellement d'effectuer les différentes étapes propres à la synthèse de circuits numériques sur FPGA. Il est alors possible d'en faire l'implémentation sur les différentes familles de puces fournies par Xilinx.

Les langages de modélisation numérique sont les langages de description matérielle de haut niveau communément appelé HDL (Hardware Description Language). Un langage HDL est une instance d'une classe de langage informatique ayant pour but la description formelle d'un système électronique. Le HDL est alors un langage de description de circuits logiques en électronique, utilisé pour la conception d'ASICs (Application-Specific Integrated Circuits) et de FPGAs (Field-Programmable Gate Array)

Le HDL peut généralement:

- décrire le fonctionnement du circuit,
- décrire sa structure,
- assurer la documentation,
- preuve formelle,
- vérification de netlist (LVS),
- et tester le circuit et le vérifier par simulation

5.2 Manipulation de l'outil ModelSim

ModelSim est un outil de **Mentor Graphics**. Il fournit un environnement complet de simulation et débogage pour les designs complexes enASIC et en FPGA. Il supporte plusieurs langages de description, dont le **Verilog**, le **System Verilog**, le **VHDL** et le **System C**.

5.2.1 La simulation

Le simulateur Model Sim nous donne accès à plusieurs fenêtres dont :

- la fenêtre **Transcript** : fenêtre d'entrée de commandes et d'affichage de messages.

- la fenêtre **Source** : source VHDL - désignation de points d'arrêt.

- la fenêtre **Structure** : présente la hiérarchie de la description VHDL

- la fenêtre **Signals** : contient la liste des signaux présents dans l'entité sélectionnée.

- la fenêtre **Wave** : affiche graphiquement les résultats de simulation.

Pour manipuler ces fenêtres, voir le menu **View**.

Avant de lancer la simulation :

a) sélectionner les signaux à visualiser :

En choisissant une entité dans la fenêtre structure, faire apparaître les signaux souhaités dans la fenêtre **Signals** puis effectuer la commande suivante :

Commande**: /Signals/Add to Waveform/Signals in region**

Les signaux présents dans la liste apparaissent dans la fenêtre **Wave**

b) lancer la simulation :

Commande : **RUN**

Le temps de simulation dépend d'une option qui est définie par la commande :

/Options/Simulation Options/Default Run Length

Une modification de la source nécessite une recompilation. La commande **/File/Restart/OK** permet de lancer une nouvelle simulation sans avoir à refaire les déclarations précédentes.

5.3 Modélisation VHDL des architectures CORDIC

5.3.1 Additionneur/soustracteur

5.3.1.1 Code VHDL

```
entity addsub is
    port (abus   : in  std_logic_vector(31 downto 0);
          bbus   : in  std_logic_vector(31 downto 0);
          obus   : out std_logic_vector(31 downto 0);
          as     : in  std_logic);          --add=1, subtract=0
end addsub;

architecture synthesis of addsub is

begin
```

```
process(as,abus,bbus)
  begin
    if as='1' then
       obus <= abus + bbus;
    else
       obus <= abus - bbus;
  end if;
end process;


end synthesis;
```

5.3.1.2 La simulation de l'additionneur/soustracteur

« as » est une commande si as= 1 , le programme réalise l'opération d'addition si as= 0 il réalise l'opération de soustraction. On charge les deux vecteurs d'entrée « abus » et « bbus » et on met as a 1 on aura le résultat suivant :

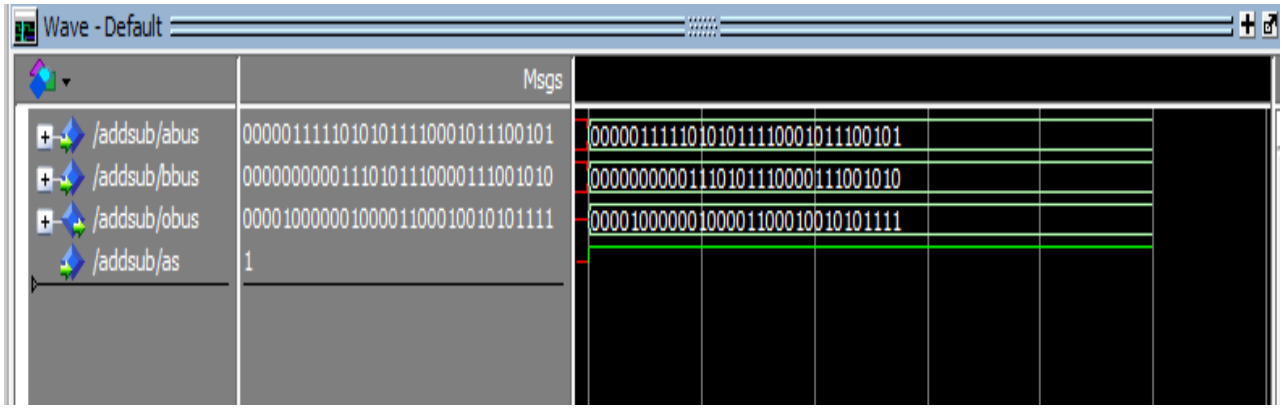

Figure 5.1 : la simulation d'un additionneur/soustracteur

5.3.2 registre à décalage

5.3.2.1 Code VHDL

```
entity shiftn is
   port (ibus   : in  std_logic_vector(31 downto 0);
         obus   : out std_logic_vector(31 downto 0);
```

```
        n      : in  std_logic_vector(4  downto 0));          --shift by n
end shiftn;
```

5.3.2.2 la simulation de registre à décalage

On charge le vecteur d'entrée « ibus » et le vecteur n qui représente le nombre de bit de décalage, si on charge par exemple n avec la séquence « 00001 » c-t-d on décale notre registre de 1 bit on aura le résultat de la simulation suivant :

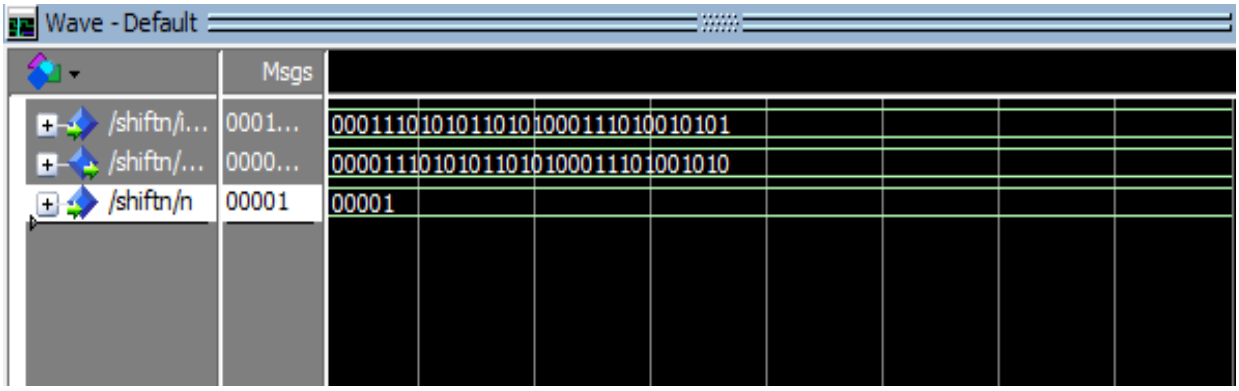

Figure 5.2 : la simulation d'un registre à décalage

5.3.3 La simulation de l'architecture Pipeline

On choisit l'angle égale a 30° sur 16 bits, on convertit la valeur qui est en degré au binaire en passant par le décimal, En utilisant la règle de trois on trouve :

$360° \rightarrow 2^{16}$

$30° \rightarrow (30*2^{16})/360 \approx 5461$ (déc) =1555 (hexa)

Donc le programme calcul les valeurs de sinus et cosinus de l'angle :

Ain =5461=0001010101010101(sur 16 bits),

Apres la simulation on trouve

Sin : 16380(déc.) = 3FFC (hexa)

Cos : 28381(déc.) = 6EDD (hexa)

Les sorties représentent des valeurs dans l'intervalle -1 à +1. Les résultats peuvent être calculés comme suit:

$2^{15} \rightarrow 1$

16380 → $(1*16380)/2^{15}$= 0,4998779296875

28381→ $(1*28381)/2^{15}$=0,866119384765625

Attendu que le résultat aurait été de 0,5 et 0,8660.

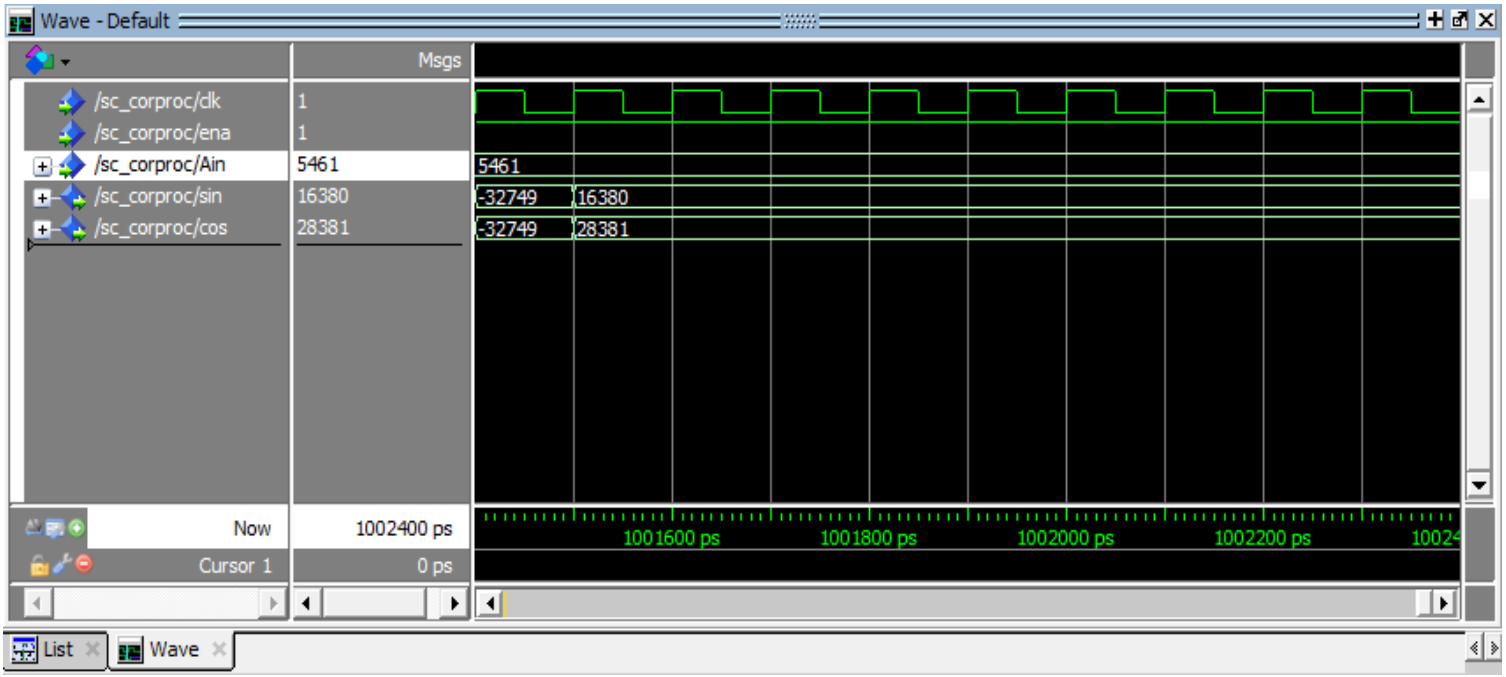

Figure5.3 Simulation de l'algorithme CORDIC ou les outputs sont Sin et Cos

Tableau 5.1: Les outputs Sin / Cos pour certains angles

	0 deg	**30 deg**	**45 deg**	**deg 60**	**90 deg**
Sin (hexa)	0x01CC	0x3FFC	0x5A82	0x6EDC	0x8000
Cos (hexa)	0x8000	0x6EDD	0x5A83	0x4000	0x01CC
Sin	0.01403	0.49998	0.70709	0.86609	1.00000
Cos	1.00000	0.86612	0.70712	0.50000	0.01403

Bien que l'algorithme soit très précis de petites erreurs peuvent être introduites par l'algorithme (voir exemple et tableau des résultats). Cela devrait être seulement un problème lors de l'utilisation du l'algorithme au cours de la toute la plage de sortie, car la différence entre +1 (0x7FFF) et -1 (0x8000) est seulement 1bit.

La figure 5.4 présente le Dataflow de l'architecture pipeline pour 16 itérations :

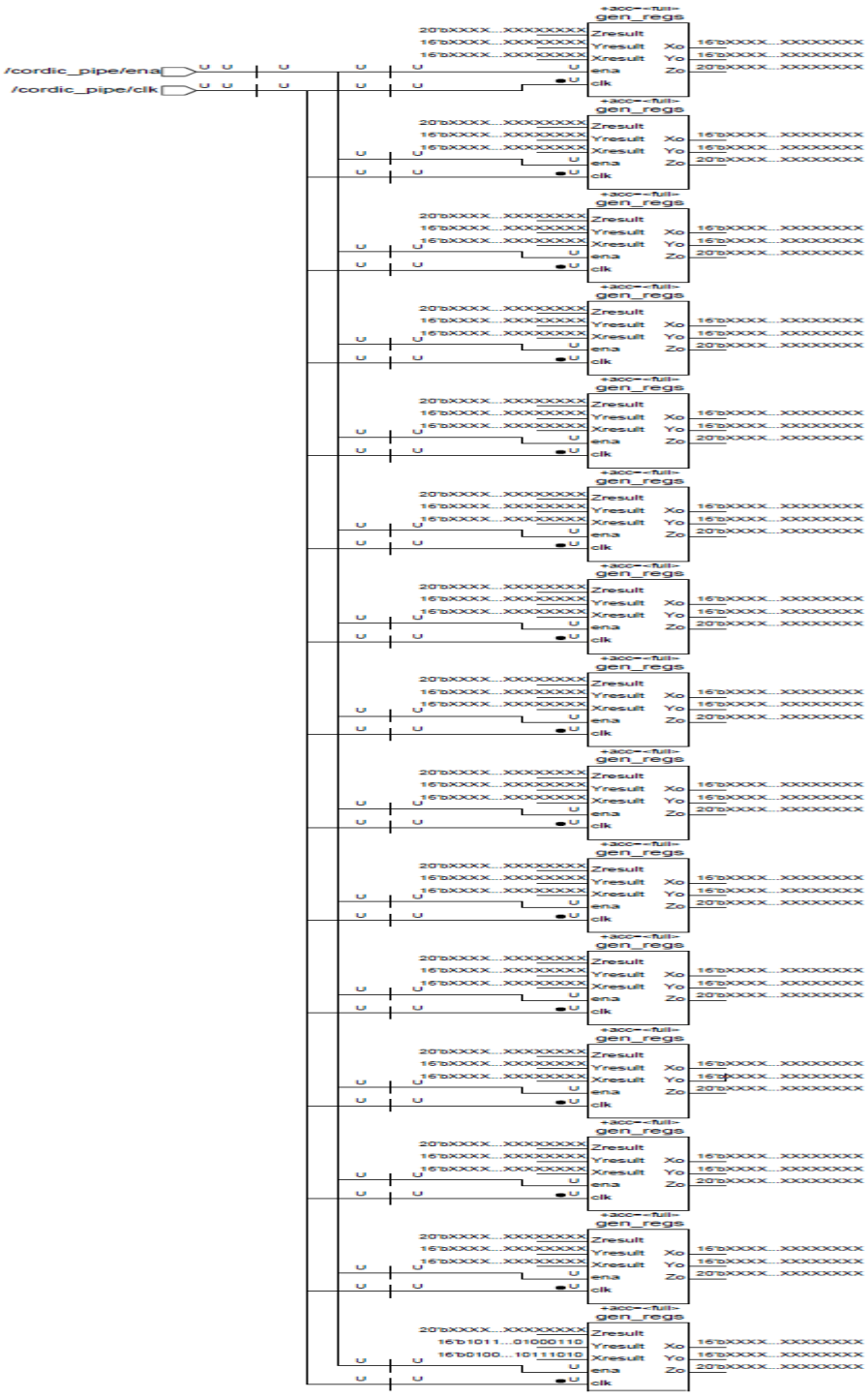

Figure 5.2 Dataflow de l'architecture pipeline

5.3.4 Exemple de simulation de l'architecture série

La simulation est donnée par la figure suivante :

Pour l'exemple de l'angle = 30° = 30*pi/180= 0,523598775598298873077107230546 58 radiant, par la conversion en binaire on trouve :(00100001100000101010010001110000)
Et en HEXADECIMAL 30°= 2182A470
Voila les résultats obtenus :

Sin=(00011111111111111111111111100001)= 0,499999971129000186920166015625
Cos=(00110111011011001111010110011101)= 0,866025355644524097442626953125

On rappel que le nombre de bits utilisés pour cette exemple c'est 32 bits ce qui implique les résultats trouvés concernant le nombre de chiffre après la virgule

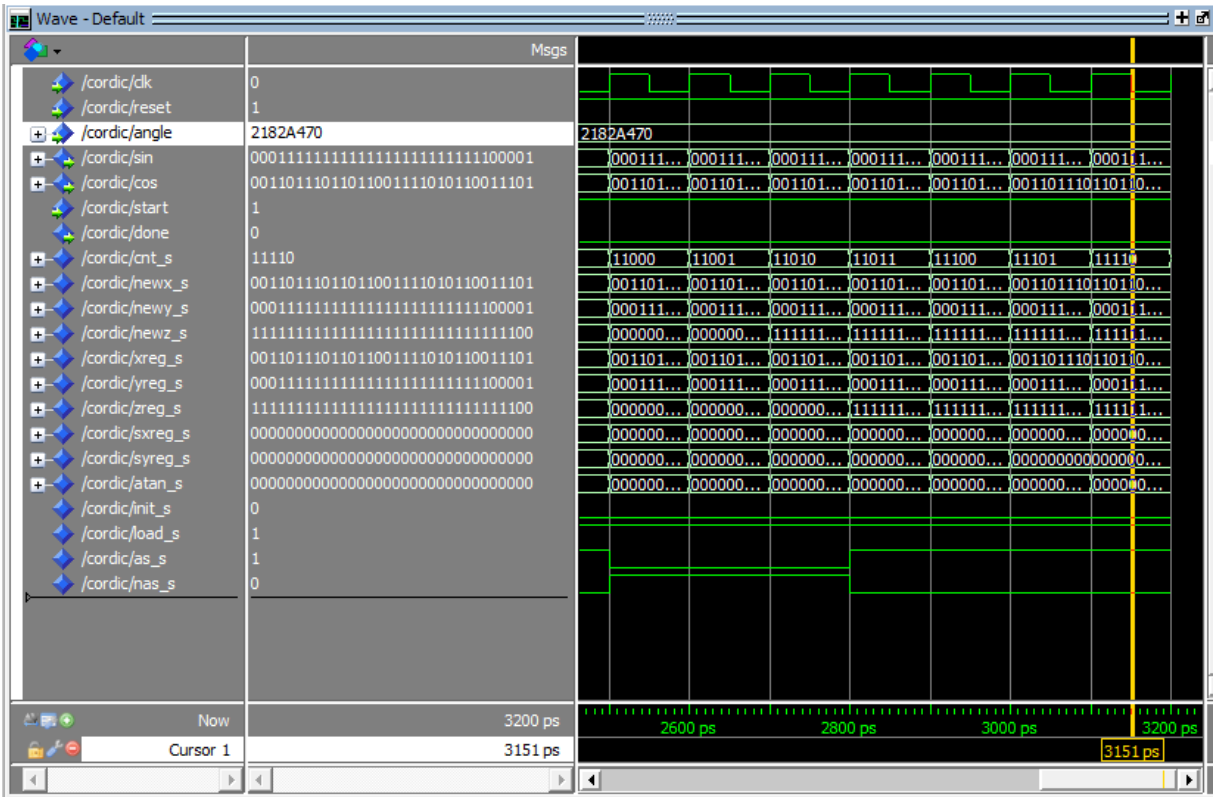

Figure5.4 Simulation de l'algorithme CORDIC ou les outputs sont Sin et Cos

5.3.5 Exemple de simulation de l'architecture parallèle

La simulation est donnée par la figure suivante :

Pour l'exemple de l'angle = 30° = 30*pi/180= 0,523598775598298873077107230546 58 radiant, par la conversion en binaire on trouve :(0010000110000010)

Voila les résultats obtenus :
Pour le mode rotation on initialise x par : 0.607252935= 0010011011011101 en binaire
Et y par (0000000000000000)
Le résultat
X_OUT=(0001111111111101)= 0,49981689453125
Y_OUT=(0011011101101111) = 0,86614990234375

On rappel que le nombre de bits utilisés pour cette exemple c'est 16 bits ce qui implique les résultats trouvés concernant le nombre de chiffre après la virgule

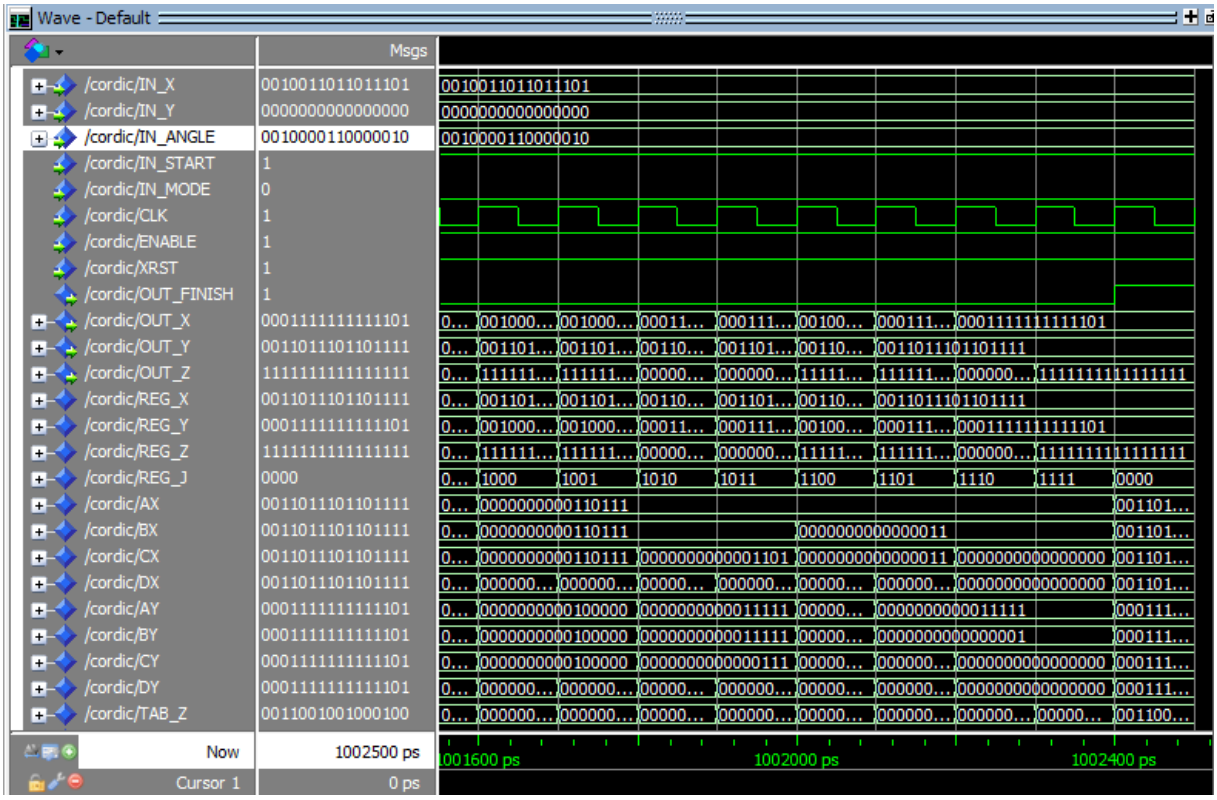

Figure5.5 Simulation de l'algorithme CORDIC ou les outputs sont Sin et Cos

Conclusion générale:

L'algorithme de CORDIC est une méthode simple et efficace pour le calcul d'une gamme de fonctions complexes. S'appuyant sur une technique décalage-addition et de rotation vectorielle, l'algorithme calcul par approximation la plupart des fonctions basées sur la trigonométrie.

L'opérateur CORDIC présente donc un très gros avantage, puisque comme nous l'avons vu, il n'utilise aucune opération arithmétique complexe pour parvenir au résultat final .La conception et l'implémentation d'un processeur CORDIC conventionnel est facilement réalisable.

Dans ce projet, un module de CORDIC est conçu et simulé à l'aide Xilinx ISE VHDL comme un outil de synthèse. La sortie du noyau CORDIC est analysée et vérifiée sur le banc d'essai, et comparées avec les valeurs réelles obtenues à partir de MATLAB.

ANNEXES

Annexe A

Comparaison entre les architectures CORDIC au niveau de circuit FPGA Xilinx

Tableau : Résumé des résultats de synthèse CORDIC basée sur FPGA Xilinx [12]

architecture	taille	Fondation Express 1.5i	dispositif	temp	Débit de données	*Fréquence*
	bits	aire/retard		ns	Msps	MHz
Itérative	12	106/139	XC4010XL-09	370.3	2.7	32.1
Itérative	14	133/145	XC4010XL-09	526.3	1.9	27.5
Itérative	16	162/178	XC4010XL-09	588.2	1.7	27.2
Itérative	24	317/376	XC4062XL-09	1643.8	0.6	14.6
Itérative	32	506/626	XC4062XL-09	2480.6	0.4	12.9
Cascade	12	210/210	XC4010XL-09	187.6	5.3	
Cascade	14	288/288	XC4010XL-09	192.9	5.2	
Cascade	16	378/378	XC4062XL-09	330.0	3.1	

Annexe B

Script MATLB :Cordic_ généralisé.m

```
% Calcul de 'cos' et 'sin' d'un angle 'theta' (en radians)
% par l'algorithme CORDIC. Résultat dans le vecteur 'v'.
% 'n' est le nombre d'itérations (la précision augmente avec lui)

v=[1;0];
d=1;
%Kn=prod(1./sqrt(1+2.^(-2*(0:(n-1)))));
Kn=0.6073;

  theta=input('donner theta')
```

```
  n=input('donner n')
if ((theta<0) && (theta>-pi/2))
     beta1 = theta + pi/2;
 choix=4;
elseif ((theta < pi/2)&&(theta >0))
 beta1=theta;
 choix=2;
elseif ((theta>pi/2)&&(theta<pi))
 beta1 = theta - pi/2;
 choix=3;
else
 beta1 = theta+pi;
 choix=1;
end

for i=0:n-1
    R=[1 -d*2^-i;d*2^-i 1];
    v=R*v;
    beta1=beta1-d*atan(2^-i);
    d=sign(beta1);
end
v=v*Kn;
if(choix==1)
sin_cos=-v;
elseif(choix==2)
sin_cos= v;
elseif(choix==3)
sin_cos=[v(2);-v(1)];
elseif(choix==4)
sin_cos=[-v(2);v(1)];
end
sin_cos
```

Annexe C

Architecture pipeline en VHDL

-- cordic_pipe.vhd

-- Calcul les valeurs de Sin et Cos

-- utilise : p2r_codic.vhd and p2r_cordicpipe.vhd

library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

entity cordic_pipeis

```
port(
clk: in std_logic;
ena: in std_logic;
 Ain:  in signed(15downto0);
sin: out signed(15downto0);
cos: out signed(15downto0));
end entity cordic_pipe;
architecture dataflow of cordic_pipe is
constant PipeLength: natural:=15;
constant P: signed(15downto0):=x"4dba";-- définir constante globale
componentp2r_cordicis
generic(
PIPELINE: integer:=15;
WIDTH: integer:=16);
port(
clk:in std_logic;
ena:in std_logic;
Xi: in signed(WIDTH-1downto0);
Yi: in signed(WIDTH-1downto0):=(others=>'0');
 Zi:in signed(WIDTH-1downto0);
Xo: out signed(WIDTH-1downto0);
Yo: out signed(WIDTH-1downto0)
);
 end component p2r_cordic;
begin
u1:p2r_cordic
generic map(PIPELINE=>PipeLength,WIDTH=>16)
```

```
 port map(clk=>clk,ena=>ena,Xi=>P,Zi=>Ain,Xo=>cos,Yo=>sin);
 end architecture dataflow;
```

-- **VHDL implementation of cordic algorithm**

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

entity p2r_cordic is
	generic(
		PIPELINE : integer := 15;
		WIDTH    : integer := 16);
	port(
		clk      : in std_logic;
		ena : in std_logic;
		Xi       : in signed(WIDTH -1 downto 0);
		Yi : in signed(WIDTH -1 downto 0) := (others => '0');
		Zi       : in signed(WIDTH -1 downto 0);
		Xo       : out signed(WIDTH -1 downto 0);
		Yo       : out signed(WIDTH -1 downto 0)
	);
end entity p2r_Cordic;


architecture dataflow of p2r_cordic is
	--	défenitions de TYPE
	type XYVector is array(PIPELINE downto 0) of signed(WIDTH -1 downto 0);
```

```
	type ZVector is array(PIPELINE downto 0) of signed(19 downto 0);
	--déclarations  des COMPONENTs
	component p2r_CordicPipe
	generic(
		WIDTH		: natural := 16;
		PIPEID		: natural := 1
	);
	port(
		clk		: in std_logic;
		ena		: in std_logic;

		Xi		: in signed(WIDTH -1 downto 0);
		Yi		: in signed(WIDTH -1 downto 0);
		Zi		: in signed(19 downto 0);

		Xo		: out signed(WIDTH -1 downto 0);
		Yo		: out signed(WIDTH -1 downto 0);
		Zo		: out signed(19 downto 0)
	);
	end component p2r_CordicPipe;
	--	SIGNALS
	signal X, Y	: XYVector;
	signal Z	: ZVector;
	--	corp de l'ACHITECTURE
begin
	X(0) <= Xi;
	Y(0) <= Yi;
```

```
        Z(0)(19 downto 4) <= Zi;
        Z(0)(3 downto 0) <= (others => '0');
        -- generer le pipeline
        gen_pipe:
        for n in 1 to PIPELINE generate
                Pipe: p2r_CordicPipe
                        generic map(WIDTH => WIDTH, PIPEID => n -1)
                                port map ( clk, ena, X(n-1), Y(n-1), Z(n-1), X(n), Y(n), Z(n) );
        end generate gen_pipe;
        -- les outputs
        Xo <= X(PIPELINE);
        Yo <= Y(PIPELINE);
end dataflow;
```

--file: p2r_CordicPipe.vhd

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

entity p2r_CordicPipe is
        generic(
                WIDTH           : natural := 16;
                PIPEID          : natural := 1
        );
        port(
                clk             : in std_logic;
                ena             : in std_logic;
```

```
        Xi          : in signed(WIDTH -1 downto 0);
        Yi          : in signed(WIDTH -1 downto 0);
        Zi          : in signed(19 downto 0);

        Xo          : out signed(WIDTH -1 downto 0);
        Yo          : out signed(WIDTH -1 downto 0);
        Zo          : out signed(19 downto 0)
    );
end entity p2r_CordicPipe;
architecture dataflow of p2r_CordicPipe is
    function CATAN(n :natural) return integer is
    variable result :integer;
    begin
        case n is
            when 0 => result := 16#020000#;
            when 1 => result := 16#012E40#;
            when 2 => result := 16#09FB4#;
            when 3 => result := 16#05111#;
            when 4 => result := 16#028B1#;
            when 5 => result := 16#0145D#;
            when 6 => result := 16#0A2F#;
            when 7 => result := 16#0518#;
            when 8 => result := 16#028C#;
            when 9 => result := 16#0146#;
            when 10 => result := 16#0A3#;
            when 11 => result := 16#051#;
            when 12 => result := 16#029#;
```

```
                when 13 => result := 16#014#;
                when 14 => result := 16#0A#;
                when 15 => result := 16#05#;
                when 16 => result := 16#03#;
                when 17 => result := 16#01#;
                when others => result := 16#0#;
        end case;
        return result;
end CATAN;

function Delta(Arg : signed; Cnt : natural) return signed is
        variable tmp : signed(Arg'range);
        constant lo : integer := Arg'high -cnt +1;
begin
        for n in Arg'high downto lo loop
                tmp(n) := Arg(Arg'high);
        end loop;
        for n in Arg'high -cnt downto 0 loop
                tmp(n) := Arg(n +cnt);
        end loop;
        return tmp;
end function Delta;

function AddSub(dataa, datab : in signed; add_sub : in std_logic) return signed is
begin
        if (add_sub = '1') then
                return dataa + datab;
```

```
            else
                return dataa - datab;
            end if;
        end;
        --      corp de l'ARCHITECTURE
        --
        signal dX, Xresult      : signed(WIDTH -1 downto 0);
        signal dY, Yresult      : signed(WIDTH -1 downto 0);
        signal atan, Zresult    : signed(19 downto 0);
        signal Zneg, Zpos       : std_logic;
begin
        dX <= Delta(Xi, PIPEID);
        dY <= Delta(Yi, PIPEID);
        atan <= conv_signed( catan(PIPEID), 20);
        Zneg <= Zi(19);
        Zpos <= not Zi(19);
        -- xadd
  Xresult <= AddSub(Xi, dY, Zneg);

        -- yadd
        Yresult <= AddSub(Yi, dX, Zpos);

        -- zadd
        Zresult <= AddSub(Zi, atan, Zneg);

        gen_regs: process(clk)
        begin
```

```
            if(clk'event and clk='1') then
                if (ena = '1') then
                    Xo <= Xresult;
                    Yo <= Yresult;
                    Zo <= Zresult;
                end if;
            end if;
        end process;
end architecture dataflow;
```

Bibliographie:

[1] Gérald ARNOULD : THÈSE DE DOCTORAT : 'Etude et Conception d'Architectures Haut-Débit pour la Modulation et la Démodulation Numériques' 2006

[2] *G.BINET : MdC 61 2007 -2008* SCHEMA GENERAL D'UN SYSTEME DE COMMUNICATION *Université de Caen - UFR de Sciences Transmission de l'information*

[3] Oussama Frioui, Fayrouz Haddad, Lakhdar Zaid, Wenceslas Rahajandraibe : Évolution des standards/architectures pour les communications sans fil Application aux systèmes multistandards en technologie CMOS

[4] Louis REYNIER: Bilan de liaison – Application SFH 534

[5] Vikas Kumar' :THÈSE DE DOCTORAT :' FPGA IMPLEMENTATION OF DFT USING CORDIC ALGORITHM'

[6] Pramod K. Meher, *Senior*, Javier Valls, , Tso-Bing Juang, K. Sridharan, , Koushik Maharatna : 50 Years of CORDIC: Algorithms, Architectures and Applications

[7] Pierre Langlois : INF3500, Conception et réalisation de systèmes numériques ; décembre 2009

[8]http://www.liafa.univparisdiderot.fr/~carton/Enseignement/Architecture/Cours/Adders/index.html

[5] ANDREA GOLDSMITH : Wireless Communication; Stanford University

[9] Rohit Kumar Jain: Design and FPGA Implementation of CORDIC-based 8-point 1D DCT Processor 2011

[12 Tanya Vladimirova et Hans Tiggeler :]FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm:

[13] Grant Hanipson. Étudient membre. IEEE Andrew Papliriski. Membre. IEEE :A VHDL Implementation of a C O R D I C Arithmetic Processor Chip

[15]Hamami : Cours de microprocesseurs spécialisés, Introduction aux circuits FPGA-5ieme année électronique-

[16] Laouar Nizar, Laraba Mohammed Seddik : Détection d'un Mouvement dans une séquence Vidéo par Filtres Morphologiques PFE Ecole Nationale Polytechnique, 2009

[17] Pramod K. Meher, Javier Valls, Tso-Bing Juang, and K. Sridharan, et Koushik Maharatna: 50 Years of CORDIC: Algorithms, Architectures, and Applications