République Algérienne Démocratique et Populaire

Ministre de l'Enseignement Supérieur et de la Recherche Scientifique

Université Dr Yahia Fares de MEDEA



Mémoire
En vue de l'obtention du diplôme de Magistère

Discipline : Electronique
Option: Instrumentation & Microélectronique

ETUDE & SIMULATION D'ARCHITECTURE DE MODULATION-DEMODULATION NUMERIQUE SUR CIRCUIT FPGA

Par

Toufik BOUCHERIT

JURY

Pr. Salam ABUDURA	Professeur, U. Médéa	Président
Dr. Noureddine BATEL	MCA, U. Médéa	Examinateur
Pr. Abderrezak GUESSOUM	Professeur, U. Blida	Examinateur
Dr. Mohamed OULDZMIRLI	MCA, U. Médéa	Rapporteur

Soutenue le 28 Novembre 2012

Remerciement

Je profite de cette occasion pour remercier tous ceux qui m'ont aidé à réaliser ce projet dans les meilleures conditions.

Je tiens à exprimer mes vifs remerciements en premier lieu à mon encadreur, Mr. Mohamed Ouldzmirli, Pour l'aide déterminante qu'il m'a accordé et pour l'intérêt qu'il a porté à mon projet. Qu'il trouve ici l'expression de mon profond respect.

Je ne manquerai pas cette occasion pour remercier mes enseignants pour leur sympathie et leur disponibilité.

Je remercie Mr Khaled Belhout pour son aide précieuse et sa collaboration durant la période de ce projet.

Enfin, je tiens à exprimer ma gratitude aux membres de jury qui ont accepté de juger ce travail.

Résumé

Ce mémoire porte sur l'adaptation au domaine tout numérique des techniques de modulation et de démodulation utilisées pour les transmissions de données. Le passage au «tout numérique» provoque de nombreux changements, à la, fois dans la conception, et dans les méthodes d'adaptation des algorithmes à l'architecture. En effet, la conversion des techniques analogiques en numérique s'avère générer de nombreuses contraintes qui brident les performances du système résultant. Aussi, ce travail définit des algorithmes, qui s'affranchissent des structures complexes présentant des chemins critiques difficiles à réduire, telles les boucles de rétroaction très souvent utilisées dans les architectures analogiques et qui ne sont pas toujours adaptées aux composants numériques. L'architecture détaillée d'un modulateur de phase et d'amplitude, et celle d'un démodulateur associé sont présentées et implantées sur un circuit FPGA. Leurs performances peuvent être supérieures à celles obtenues par simple transposition des architectures analogiques existantes sur un circuit numérique. Elles présentent en outre, de bonnes capacités d'évolution.

Mots-clés: Architectures numériques haut débit, modulation et démodulation totalement numériques, CORDIC, modélisation RTL.

Abstract

The aim of this thesis is to adapt the modulation and demodulation techniques used in modern data transmission. The main issues are important modifications that must be incorporated to the design and the methods of implementing algorithms in hardware. This is because the analogue to digital conversion usually imposes several constraints that worsen performance of the resulting system. In this work are proposed algorithms that allow avoiding using feedback loops that usually appear in purely analogue architectures, and which are not suitable for digital implementations. The detailed architectures for a phase and amplitude modulator and for the corresponding demodulator are presented and implemented on FPGA. These new digital circuits significantly outperform their existing digital counterparts which could be obtained by simple analogue to digital conversion.

Keywords: High-throughput digital architectures, all-digital modulation and demodulation, CORDIC, RTL modelisation.

الملخَّـــص

إن الهدف من هذه المذكرة هو تبني تقنيات التعديل وكشـــف التعـــديل الـــرقميين واستعمالها لإرسال واستقبال المعلومات بطريقة حديثة، و لأن التحويـــل مـــن النظـــام التناظري الى الرقمي يفرض عدة قيود التي تحد من أداء هذه التقنيات قمنا ببعض التغييرات التي يجب ان تدمج في تصميم الخوارزميات المتعلقة بهذه الأخيرة وتطبيقهـــا علـــى أرض الواقع. في هذا العمل نقترح بعض الخوارزميات التي تتجنب استعمال حلقات العودة التي نراها عادة في الانظمة التناظرية ولا تناسب الانظمة الرقمية. الهندسة التفصيلة للمعـــدل الساعي ونفس المعدل في الصفحة وكاشف التعديل الذي يقابله يمكن تطبيقها في الـــدارة ذات البوابات القابلة للبرمجة، في هذه الحالة يتميز المعدل وكاشفه بالأداء العالي مقارنة بما يكونان عليه في الحالة التناظرية.

كلمات دلالية: هندسة رقمية ذات التدفق العالي، التعديل وكشف التعديل الـــرقميين كلية،حاسبة مقدار الدوران الرقمية، التجسيم على مستوى ناقل السجل.

References

[1] P.J. Graumann, L. e. Turner: Implementing Digital Signal Processing Algorihtms using Pipelined Bit-Serial Arithmetic and FPGA, First International ACM/SIGDA Workshop on FPGA, 1992.

[2] Ph. Fraisse, R. Protière, D. Marty-Dessus, *Télécommunications 1 : Transmission de l'information*, Ellipses, Paris, 1999.

[3] G. Pujolles, *Les réseaux*, 3ème édition, Eyrolles, Paris, 2003.

[4] M. Stein, *Les modems pour la transmission de données numériques*, 2ème édition, Masson, Paris, 1991.

[5] A. Glavieux, M. Joindot, *Communications numériques*, Masson, Paris, 1996.

[6] M. Bellanger, *Traitement numérique du signal – théorie et pratique*, 7ème édition, Masson, Paris, 2002.

[7] IEEE standard 802.11, "Wireless LAN medium access control and physical layer specifications", IEEE Computer Society, August 1999.

[8] E. Grass *et al.*, "On the single-chip implementation of a HiperLan/2 and IEEE 802.11a capable modem", *IEEE Personal Communications Mag*, vol. 8, pp. 48–57, Dec. 2001.

[9] C. E. Shannon, "A mathematical theory of communication", *Bell System Technical J.*, vol. 27, pp. 379–423, 623–656, July, October 1948.

[10] OFDM-based 802.16.3 sub-11 GHzBWA Air Interface Physical Layer proposal, IEEE 802.16 BroadbandWireless AccessWorking Group Oct. 2000; http ://ieee802.org/16.

[11] P. Brenner, "A technical tutorial on the IEEE 802.11 protocol", BreezecomWireless Communications, July 1996.

[12] J. Tierney, C. M. Rader, and B. Gold, "A digital frequency synthesizer",*IEEE Trans. Audio Electroacoust.*,vol. 19, pp. 48–57, Mar.1971.

[13] *Virtex-II™ Platform FPGAs:* Complete Data Sheet, Xilinx Product

Specification, Mars 2005.www.xilinx.com/literature/datasheet/ds031.pdf

[14] R. Andraka, "A survey of cordic algorithms for FPGA based computers", in *Proc. Sixth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, ACM/SIGDA, ACM, 1998.

[15] J. Vankka, "Methods of mapping from phase to sine amplitude in direct digital synthesis", *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 44, no. 2, pp. 526–534, March 1997.

[16] A. M. Sodagar and G. R. Lahiji, "A pipelined ROM-less architecture for sine-output direct digital frequency synthesizers using the second-order

[17] D. De Caro, E. Napoli, and A. G. M. Strollo, "ROM-less direct digital frequency synthesizers exploiting polynomial approximation", in *Proc. 9th Int. Conf. on Electronics, Circuits and Systems (ICECS 2002)*, Sept. 2002, pp. 481–484.

[18] J. M. P. Langlois and D. Al-Khalili, "Hardware optimized direct digital frequency synthesizer architecture with 60 dBc spectral purity", in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS 2002)*, vol. 5, Kingston, Ontario, Canada, May 2002, pp. V-361–V-364.

[19] S.-I. Liu, T.-B. Yu, and H.-W. Tsao, "Pipeline direct digital frequency synthesizer using decomposition method", *IEE Proc. – Circuits, Devices and Systems*, vol. 148, pp. 141–144, March 2001.

[20] H. T. Nicholas, H. Samueli, and B. Kim, "The optimization of direct digital frequency synthesizer performance in the presence of finite word length effects", in *Proc. 42nd Annual Frequency Control Symposium*, 1998, pp. 357–363.

[21] J. E. Volder, "The CORDIC trigonometric computing technique", *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.

[22] J. E. Volder, "The birth of CORDIC," *J. VLSI Signal Processing*, vol. 25, no. 2, pp. 101–105, June 2000.

[23] J. S. Walther, "A unified algorithm for elementary functions", in *Proc. AFIPS Spring Joint Comput. Conf.*, vol. 38, Atlantic City, NJ, USA, May 1971, pp. 379–385.

[24] G. Hampson and Paplinski, "A VHDL implementation of a CORDIC arithmetic processor chip", Tech. Rep. 94-9, Monash Univ., Clayton, Victoria, Australia, 10 Oct. 1994.

[25] O. Mencer, L. Séméria, M. Morf, and J.-M. Delosme, "Application of reconfigurable CORDIC architectures," *J. VLSI Signal Processing*, vol. 24, no. 2–3, pp. 211–221, Mars 2000.

[26] S. Nahm and W. Sung, "A fast direction sequence generation method for CORDIC processors", in *Proc. IEEE Int. Conf. on Acoustics,*

[27] Y. Hen Hu, "CORDIC-based VLSI architectures for digital signal processing", *IEEE Signal Processing Magazine*, pp. 16–35, Juil. 1992.

[28] J. Duprat and J.Muller, "The CORDIC algorithm : new results for fast VLSI implementation", *IEEE Trans. Comput.*, vol. C-42, pp. 168–178, Feb. 1993.

[29] C. Y. Kang and E. E, Swartzlander, Jr., "A digit-pipelined direct digital frequency synthesis architecture", in *Proc. IEEE Workshop on Signal Processing Systems (SIPS 2003),* Aug. 2003, pp. 224–229.

[30] S. Lin and D.J. Costello, Jr., *Error Control Coding : Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1983.

[31] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[32] P. H. R. Scholefield, "Shift registers generating maximum length sequences", *Electronic Technology*, vol. 37, pp. 389–394, Oct. 1960.

[33] D. B. Thomas and W. Luk, "High quality uniform random number generation for massively parallel simulations in FPGAs", in *Proc. 2005 Int. Conf. Reconfigurable Computing and FPGAs (ReConfig'05)*, Puebla City, Mexico, 28-30 Sept 2005, pp. 84–91.

[34] M. Goresky and A. Klapper, "Fibonacci and Galois representation of feedback with carry shift registers", *IEEE Trans. Inform. Theory*, vol. 48, pp. 2816–2836, Nov. 2002.

[35] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*,

[36] P. Alfke, "Efficient shift register, LFSR counters, and long pseudo-random sequence generators", Application Note, Xilinx, 1995. Cambridge University Press, Cambridge, UK, 1986.

[37] E. J. Watson, "Primitive polynomials (mod 2)", *Math. Comp.*, vol. 16, pp. 368—369, 1962,

[38] S. W. Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, CA, 1982.

[39] J. L. Massey, "Shift-register synthesis and BCH decoding", *IEEE Trans. Inform. Theory*, vol. IT-15, no. 1, pp. 122–127, Jan. 1969.

[40] F. Arnault, T. P. Berger, and A. Necer, "Feedback with carry shift registers synthesis with the Euclidean algorithm", *IEEE Trans. Inform. Theory*, vol. 50, no. 5, pp. 910–917, May 2004.

[41] M. R. Stan, "Synchronous up/down counter with clock period independent of counter size", in *Proc. 13th IEEE Symp. Computer Arithmetic (ARITH-13)*, 1997, p. 274–281. vol. 1, April 1997, pp. 635–638.

[41] Y. Song and B. Kim, "Quadrature direct digital frequency synthesizers using interpolation-based angle rotation", *IEEE Trans. VLSI Syst.*, vol. 12, pp.701–710, July 2004.

Table des Matières

Table des matières……...II

Liste des tableaux ……..V

Liste des figures ……VI

Introduction générale ……1

Chapitre 1 : TECHNIQUE DE TRANSMISSION NUMERIQUE……4

1.1. Introduction…… 5

1.2. Chaîne de communication numérique……. 5

1.2.1. Transmission réseau…….. 6

1.2.2. Structuration fonctionnelle de la couche physique…….10

1.2.3. Structuration fonctionnelle de la couche physique à tous et contraintes d'un traitement numérique……...13

1.3. Transmission d'un signal numérique en bande transposée……..15

1.3.1. Échantillonnage des signaux…….. 15

1.3.2. Cas des signaux sinusoïdaux…….. 16

1.3.3. Quantification d'un signal…….. 17

1.3.4. Production numérique de signaux en bande transposée……. 18

1.4. Architectures analogiques pour la modulation et la démodulation Numériques ……..19

1.4.1. Modulateur QAM analogique…….19

1.4.2. Démodulateur QAM analogique……..20

1.4.3. Incompatibilités avec le traitement numérique……21

1.5. Circuits FPGA dans le cadre de la modulation et la démodulation ……22

1.5.1. Technologie FPGA de Xilinx ……23

1.5.2. FPGA Virtex-II ……..….. 27

1.5.3. Limitations et avantages pour le traitement numérique du signal ……. 30
1.6. Conclusion ……31

Chapitre 2 : ARCHITECTURES NUMERIQUES POUR LA MODULATION NUMERIQUE ……. 33

2.1. Introduction ……..… 34
2.2. Architecture d'un modulateur numérique ……35
2.2.1. Les modulations numériques …… 35
2.2.2. Structure générale d'un modulateur numérique ……...….37
2.3. Codage des informations ……..40
2.3.1. Conception du pré-codeur ……...41
2.3.2. Conception du codeur ……43
2.4. Générateurs de sinusoïdes : architectures existantes…….44
2.4.1. Utilisation de mémoires et d'échantillons pré-calculés……...45
2.4.2. Utilisation couplée de filtres interpolateurs ……...47
2.5. Processeur CORDIC modifié……....49
2.5.1. Principe du processeur CORDIC ……50
2.5.2. Application à la génération paramétrique de sinusoïdes ……53
2.5.3. Architecture CORDIC itérative ……..57
2.5.4. Architecture CORDIC pipeline ……..58
2.5.5. Approximation de la valeur des $arctan2^{-i}$ …….…..61
2.6. Utilisation de la structure à double branche du pipeline……..64
2.7. Conclusion ……...65

Chapitre 3 : ARCHITECTURE NUMERIQUE POUR LA DEMODULATION NUMERIQUE ……...… 66

3.1. Introduction ……....67
3.2. Registres à décalage à rétroaction linéaire……...68

3.2.1. Principe mathématique…… 69
3.2.2. Périodicité, polynôme minimal et polynôme primitif ……70
3.2.3. Algorithme de Berlekamp-Massey ……....71
3.2.4. Architectures matérielles pour l'implantation de LFSR……...72
3.2.5. Compteurs à base de LFSR……...73
3.2.6. Décompteur apparié ……....74
3.3. Application à la démodulation entièrement numérique de la phase ……75
3.3.1. Architecture du démodulateur ……77
3.3.2. Mesure du déphasage ……...80
3.4. Conclusion …….....83

Chapitre 4 : SIMULATION & SYNTHESE…….….84

4.1. Introduction ……....85
4.2. Résultats de simulation de modulateur/démodulateur……..……....86
4.3. Résultats de synthèse de modulateur/démodulateur…….…....91
4.4. Conclusion ……94

Conclusion générale ……..95

Annexes ……96
Annexe A ……....97
Annexe B……99
Annexe C……....114
Annexe D……....116

Références ……118
Liste des acronymes ……124

Liste Des Tableaux

TAB. 1.1− *Comparaison des FPGA par rapport aux autres architectures de traitement du signal*……23

TAB.1.2 − *Ressources du circuit VIRTEX-II* ……28

TAB 4.1 − *Récapitulation des résultats de synthèse*……93

Liste Des Figures

FIG. 1.1– *Schéma simplifie d'un système de transmission numérique……....6*
FIG. 1.2– *Les couches, les protocoles, et les interfaces d'un réseau……..…7*
FIG. 1.3– *Modèle de Référence OSI……8*
FIG. 1.4– *Représentation fonctionnelle des couches physique et liaison d'un système de transmission comprenant une paire émetteur-récepteur……...11*
FIG.1.5– *Ligne de transmission, séparation analogique numérique……12*
FIG. 1.6– *Architecture d'un modulateur analogique QAM……19*
FIG.1.7- *Architecture d'un démodulateur analogique QAM……21*
FIG.1.8 – *Organisation des FPGA de Xilinx……24*
FIG.1.9 – *Architecture simplifiée d'un « Slice »……25*
FIG.1.10 – *Trois modes de configuration des LUT……..25*
FIG.1.11 – *Configuration des blocs d'E/S……27*
FIG.1.12 – *Détail de l'architecture d'un composant Virtex-II……29*
FIG. 2.1 – *Chaîne de modulation numérique QPSK-QAM* ……37
FIG. 2.2 – *Architecture de l'accumulateur de phase……38*
FIG. 2.3 – *Architectures interne de pré-codeur/codeur……40*
FIG. 2.4 – *Constellations QAM 256, QAM64, QAM16, et QPSK 4……41*
FIG. 2.5 – *Conversion des trames au format QAM 256……42*
FIG. 2.6 – *Architecture général d'un générateur de sinusoïdes à base de ROM……45*
FIG. 2.7 – *La rotation θ sur un cercle trigonométrique……51*
FIG. 2.8 – *Décomposition en plusieurs micro-rotations……52*
FIG. 2.9 – *Cellule élémentaire d'un processeur CORDIC……56*
FIG. 2.10 – *Générateur de sinusoïdes à base de processeur CORDIC itératif……..57*
FIG. 2.11 – *Générateur de sinusoïdes à base de processeur CORDIC pipeline……59*
FIG. 2.12 – *Deux architectures de décaleurs……60*

FIG. 3.1 – *Les deux architectures de LFSR parmi d'autres*……68
FIG. 3.2 – *Principe d'un compteur basé sur un LFSR*……73
FIG. 3.3 – *Mise en forme d'un signal M-PSK*……76
FIG. 3.4 – *Durées des états haut et bas pour un signal non déphasé, et un signal avec un déphasage*……77
FIG. 3.5 – *Architecture du démodulateur de phase*……78
FIG. 3.6 – *Détection d'un déphasage - premier chronogramme*……80
FIG. 3.7 – *Détection d'un déphasage - second chronogramme*……81
FIG. 3.8 – *Détection d'un déphasage - troisième chronogramme*……82
FIG. 4.1 – *La carte FPGA Virtex II XC2V1000*……85
FIG. 4.2 – *Simulation de l'étage «Pré-codeur»*……87
FIG. 4.3 – *Simulation de l'étage «Codeur»*……87
FIG. 4.4 – *Simulation de processeur CORDIC*……88
FIG. 4.5 – *Simulation de l'étage «Bloc d'échantillonnage»*……89
FIG. 4.6 – *Simulation de démodulateur*……90
FIG. 4.7 – *Schéma RTL externe du bloc 'modulateur'*……92
FIG. 4.8 – *Schéma RTL externe du bloc 'démodulateur'*……92

×

Introduction générale

Le domaine des télécommunications et des réseaux a connu de profonds changements, et une évolution rapide pendant les années 1990, avec par exemple le développement des systèmes informatiques grand public. Sur le plan technologique en particulier, les réseaux de transmission ont vu leur capacité s'accroître, notamment concernant le débit supporté. Mais la demande est toujours plus importante, et suit l'évolution des contenus transportés.

Concernant les télécommunications, différents niveaux de traitement peuvent être considérés. Entre le niveau applicatif, qui concerne directement l'utilisateur, et le médium physique de transmission, pour lequel seuls des signaux et les composantes élémentaires des données sont pris en compte, les algorithmes mis en œuvre n'ont pas les mêmes objectifs, ni les mêmes contraintes de performances. À l'extrémité du dispositif de télécommunication se trouvent les couches les plus basses de la chaîne de transmission, celles qui sont chargées de produire un signal représentatif des données à transmettre et adapté au médium, permettant sa transmission vers son destinataire. Ce sont ces couches basses qui sont les plus sensibles aux changements de protocoles de transmission, car elles doivent produire un signal dont les caractéristiques physiques et temporelles sont très dépendantes des spécifications du canal de communication. Aussi, ces éléments requièrent pour leur conception, une attention toute particulière afin de ne pas brider la vitesse de traitement qu'ils sont capables d'atteindre, ce qui limiterait leur champ d'application et les possibilités d'évolution.

Des systèmes complexes de codage et d'adaptation des signaux sont donc apparus pour exploiter au maximum les capacités des média de transmission, et les architectures de modulation et de démodulation qui en découlent ont elles aussi dû évoluer pour supporter les cadences de traitement plus élevées.

Habituellement, ces architectures reposent sur des composants électroniques analogiques compatibles avec les hautes fréquences de fonctionnement. Mais ces technologies sont chères et peu adaptées à l'évolution des normes, des algorithmes ou des protocoles utilisés. La démocratisation des circuits programmables de type FPGA[1] et leur adéquation aux petites séries ainsi que l'augmentation de leurs performances font qu'ils ne sont plus seulement utilisés à des fins de prototypage avant implantation sur silicium, mais aussi comme cible finale d'une chaîne de conception. Ils présentent de nombreux avantages, en particulier leur faible coût mais aussi le fait d'offrir une capacité d'évolution importante aux systèmes, permettant par conséquent de s'adapter rapidement aux changements de protocoles fréquents dans le domaine des télécommunications. En outre, ils s'intègrent parfaitement dans la chaîne de conception d'un système, où la réutilisation de blocs fonctionnels devient primordiale, avec l'augmentation de la complexité de ceux-ci, et des coûts et temps de développement inhérents

Dans ce mémoire, on étudie deux architectures compatibles avec l'évolution des débits et des fonctionnalités des protocoles actuels. L'une concerne la modulation, l'autre la démodulation de signaux. L'architecture pour la modulation est basée sur une version modifiée de l'algorithme CORDIC[2], originellement conçu pour la génération itérative de fonctions trigonométriques [VOLD59, VOLD00]. L'architecture de démodulation utilise des registres à décalage à rétroaction linéaire (LFSR[3]) afin de directement mesurer la phase des signaux qui lui sont fournis. Cette architecture implémentée sur un FPGA de type Virtex XC2V1000 de Xilinx à la fréquence de 300MHz.

Le premier chapitre pose les bases et fondements d'un système de transmission, et présente son architecture générale, en insistant sur les spécificités des systèmes purement numériques. Ensuite, il met en lumière les principes fondamentaux du traitement du signal, comme le théorème de l'échantillonnage qui influe sur notre

[1] *Field Programmable Gate Array*
[2] *COrdinate Rotation DIgital Computer*
[3] *Linear Feedback Shift Register*

méthodologie de conception d'une architecture entièrement numérique. Nous présentons également les solutions les plus utilisées dans les architectures analogiques, en nous focalisant sur les difficultés de leur transposition vers des systèmes entièrement numériques, afin d'extraire une nouvelle méthode pour nous affranchir des limitations introduites par la transposition des architectures analogiques sur circuit numérique.

Le second chapitre porte sur la réalisation d'une architecture purement numérique pour un modulateur. Les différentes méthodes permettant de générer un signal modulé y seront présentées, et deux d'entre elles seront plus particulièrement au centre de nos attentions. Une première méthode, utilisant des échantillons de sinusoïde pré-calculés, est souvent employée car très simple à mettre en œuvre. Une version hybride, mettant en œuvre des filtres numériques interpolateurs est également examinée. Enfin, notre nouvelle architecture construite sur la base de l'algorithme CORDIC est présentée et confrontée aux deux précédentes.

Parmi les différentes architectures de démodulation les systèmes utilisant des boucles à verrouillage de phase (PLL[1]) sont traditionnellement employés dans le monde analogique. Par conséquent je propose dans le troisième chapitre une nouvelle architecture parallèle utilisant plusieurs bancs de registres à décalage.

Le quatrième chapitre concerne la partie pratique quant en utilisant le logiciel Xilinx ISE qui permettre de simuler et de synthétiser puis assurer la validité de chaque bloc de modulation et le démodulation. Leur implantation sur circuit FPGA de type XC2V1000 permettre en suit d'évaluer leurs performances.

[1] *Phase-Locked Loop*

Chapitre 1

Technique de Transmission Numerique

1.1. Introduction

Ce chapitre présente en premier lieu une chaîne de transmission de signaux numériques, afin de situer et préciser l'importance des fonctions de modulation et de démodulation, replacées dans leur contexte. Ces deux éléments se situent à des positions déterminantes (aux deux extrémités d'un canal de transmission) ce qui implique qu'ils sont les plus sujets à des évolutions importantes lorsque le débit imposé par les protocoles de communication ou la fréquence du signal transportant les informations augmentent.

Pour comprendre et aborder les difficultés que doivent surmonter les architectures numériques lors de la production d'un signal à haute fréquence, les principes régissant la conversion entre le monde numérique et le monde analogique, tels que le théorème de l'échantillonnage, sont rappelés. De là sont déduits les critères constituant les contraintes de conception pour des modulateurs ou démodulateurs numériques équivalents à leurs contreparties analogiques. Les avantages des architectures *tout numériques* pour les systèmes de transmission sont mis en avant en particulier en termes d'évolution, de coût et de ressources. À cet égard, l'architecture générale et le fonctionnement d'un circuit de type FPGA sont analysés, ainsi que leur adéquation avec les algorithmes utilisés pour la modulation et la démodulation de signaux.

1.2. Chaîne de communication numérique

Les systèmes de transmission numérique véhiculent de l'information entre une source et un destinataire en utilisant un support physique comme le câble, la fibre optique [1]. Les signaux transportés peuvent être soit directement d'origine numérique comme dans les réseaux de données, soit d'origine analogique (parole, image...) mais convertis sous une forme numérique.

La tâche du système de transmission est d'acheminer le signal de la source vers le destinataire avec le plus de fiabilité possible. Le schéma synoptique d'un système de transmission numérique est donné à la Figure 1.1, où l'on se limite aux fonctions de base.

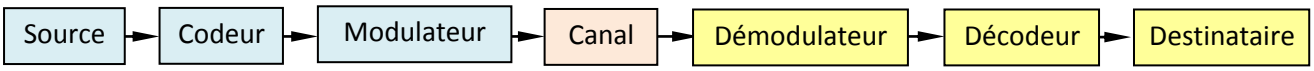


FIG. 1.1- *Schéma simplifié d'un système de transmission numérique*

1.2.1. Transmission réseau

Pour réduire la complexité de conception, la plupart des réseaux sont organisés en un empilement des couches ou des niveaux, chacun construit sur celui-de-dessous.

Le nombre de couches, le nom, le contenu, et la fonction de chaque couche diffèrent d'un réseau à réseau. Le but de chaque couche est d'offrir certains services aux couches supérieures. La figure 1.2 illustre un réseau de cinq-couches. Les entités qui composent les couches correspondant à différentes machines sont appelées pairs. Les pairs peuvent être des procédés, de périphériques ou même des êtres humains. En d'autres termes, il est des pairs, qui communiquent en utilisant le protocole entre chaque paire de couches adjacentes il existe une interface qui définit les opérations primitives et les services de la couche inférieure que met à la disposition du haut.

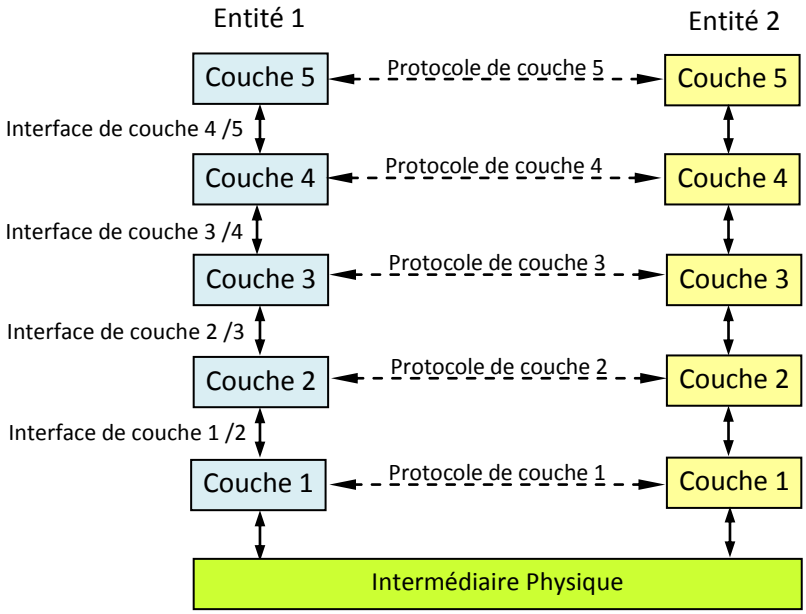


FIG. 1.2 - *Les couches, les protocoles, et les interfaces d'un réseau*

En réalité, les données ne sont transférées directement à partir de n couches sur une machine à une autre machine équivalente.

Au lieu de cela, chaque couche passe les données et informations de contrôle à la couche immédiatement au-dessous, jusqu'à ce que la couche la plus basse soit atteinte. Au dessous de le couche, 1 il existe le médium physique dans lequel la communication a lieu.

L'organisation de réseau dans la figure 1.2 est obtenue par effectuer des efforts de modélisation pour permettre de séparer les différents niveaux de fonctionnalité d'un système de traitement numérique de l'information, plus particulièrement dans le cadre de transmissions réseau. Il existe trois principaux modèles:

– le modèle OSI[1] est souvent employé pour décrire cette segmentation, bien que peu de protocoles réels soient effectivement basés sur lui.

[1] *Open System Interconnection*
[2] *Transmission Control Protocol / Internet Protocol*

– le modèle TCP/IP[2], antérieur au précédent, dispose d'une architecture modulaire ressemblant à celle d'OSI. Il est à la base du réseau mondial Internet;

– enfin, le modèle UIT-T[1] est né pour satisfaire aux besoins spécifiques des réseaux à haut-débit. Il est plus particulièrement utilisé dans les réseaux de nouvelle génération.

Bien que les protocoles liés au modèle OSI soient rarement employés plus, le modèle lui même est réellement tout à fait général et encore valide, et les dispositifs discutés à chaque couche sont toujours très importants, pour cette raison nous regarderons ce modèle en détail.

Il se compose de sept couches distinctes, et est présenté sur la figure 1.3.

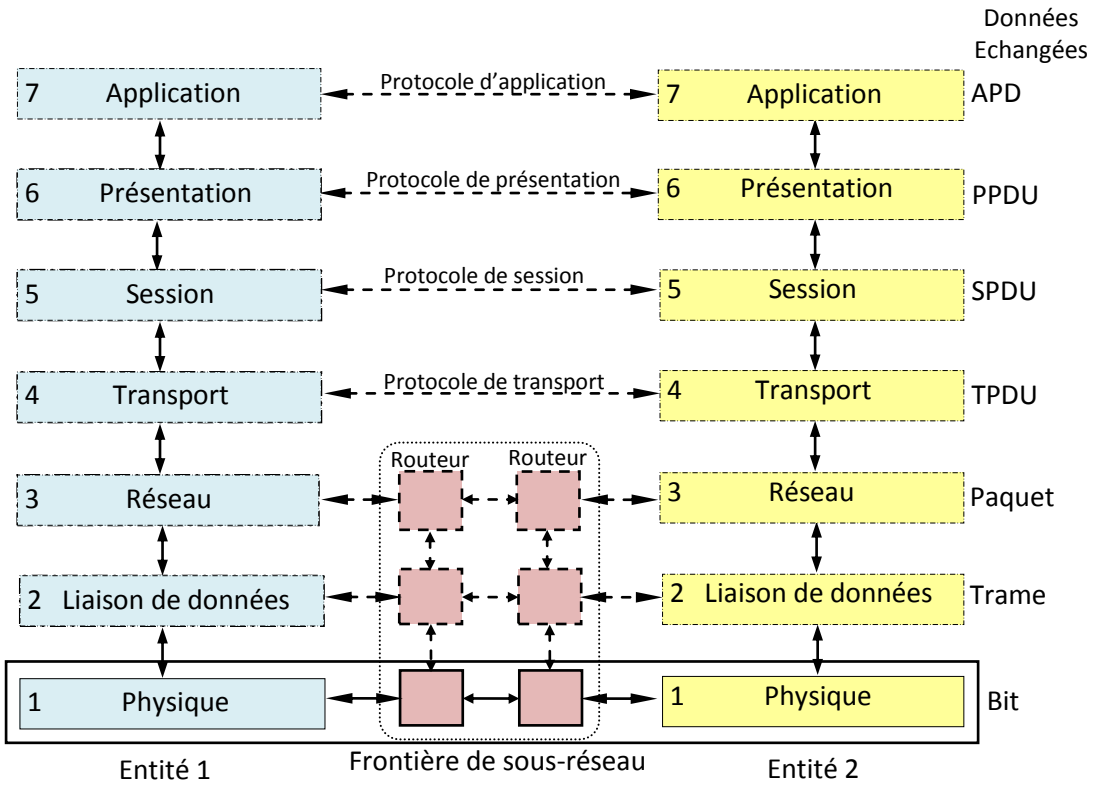


FIG. 1.3 – *Modèle de Référence OSI*

Le modèle OSI décrit des niveaux de transmission mais non les protocoles proprement dits. Il divise l'ensemble des protocoles en sept couches indépendantes

[1] *Union Internationale des Télécommunications*

entre lesquelles sont définis deux types de relations : les relations verticales entre les couches d'un même système (interfaces) et les relations horizontales relatives au dialogue entre deux couches de même niveau (les protocoles). Les couches 1, 2, 3 et 4 sont orientées transmission et les couches 5, 6 et 7 sont orientées traitement.

– La couche *application* (7) fournit les services et interfaces de communication aux utilisateurs ;

– La couche *présentation* (6) s'occupe de la syntaxe des données. Cette couche permet à deux machines de communiquer même lorsqu'elles utilisent des représentations de données différentes. Elle gère des structures de données haut niveau pour accomplir cette tâche ;

– La couche *session* (5) permet d'établir une connexion logique entre deux applications. Elle assure l'organisation et la synchronisation du dialogue ;

– La couche *transport* (4) permet l'établissement, le maintien et la rupture des connexions. L'une des tâches principales de cette couche est d'accepter des données de la couche supérieure et de les diviser en unités plus petites : il s'agit de l'opération de fragmentation. Elle offre un service réel de bout en bout de la source à la destination, indépendant du chemin effectif utilisé entre les machines.

Dans les couches plus basses, les protocoles établissent des relations entre une machine et ses voisins immédiats et non entre les machines source et destination.

En effet, les couches 1 à 3 sont chaînées alors que les couches 4 à 7 sont de bout en bout;

– La couche *réseau* (3) assure la commutation et le routage des paquets entre les nœuds du réseau. Pour le modèle OSI, il existe deux méthodes principales d'acheminement: la commutation de circuits et la commutation de paquets. C'est cette couche qui gère les nœuds du réseau ;

– La couche *liaison* (2) a pour but de transmettre les données sans erreur. Elle décompose les données de l'émetteur en trames de données puis les envoie de façon séquentielle. Différentes méthodes permettant de protéger les données contre les erreurs sont utilisées, comme le codage de détection et de correction

d'erreur. Dans les réseaux à canal partagé, cette couche s'occupe aussi de contrôler l'accès au canal par l'intermédiaire d'une sous-couche MAC[1];

– La couche *physique* (1) s'occupe de la connexion physique sur le réseau. Elle se charge de la transmission de bits à l'état brut sur un canal de transmission. Les problèmes de conception de cette couche concernent les interfaces électriques, la synchronisation, ainsi que les spécifications du support physique de transmission. En mode émission, elle est chargée de s'assurer des bonnes transmissions de données fournies par la couche liaison qui la précède sur le canal de transmission.

En mode réception, l'intégrité des données reçues est prise en compte. La gestion d'éventuels mécanismes de synchronisation et de communications bidirectionnelles est dévolue à la couche physique, le cas échéant. La transition entre la représentation des informations sous forme de signal, puis sous forme de bits, est effectuée dans cette couche. C'est à ce niveau de fonctionnement que nous nous intéressons plus particulièrement dans la suite, car c'est dans celui-ci qu'interviennent en particulier les opérations de modulation et de démodulation. Le modèle d'abstraction OSI permet donc de situer les opérations de modulation et de démodulation dans un système de transmission d'information pouvant éventuellement s'inscrire dans le cadre plus global d'un réseau à grande échelle. Les étapes de traitement effectuées dans cette couche physique sont exposées dans la section 1.2.3.

1.2.2. Structuration fonctionnelle de la couche physique

Les fonctions de base d'un système de transmission numérique sont représentées par la figure 1.4, ces fonctions remplissant le rôle d'interface entre le signal continu adapté au médium de transmission et les informations binaires transmises aux couches supérieures. Cette figure présente à la fois des fonctions appartenant à la couche

[1] *Medium Access Control*

physique, et d'autres à la couche liaison de données. Mais certaines, comme le codage de canal peuvent appartenir aux deux niveaux simultanément.

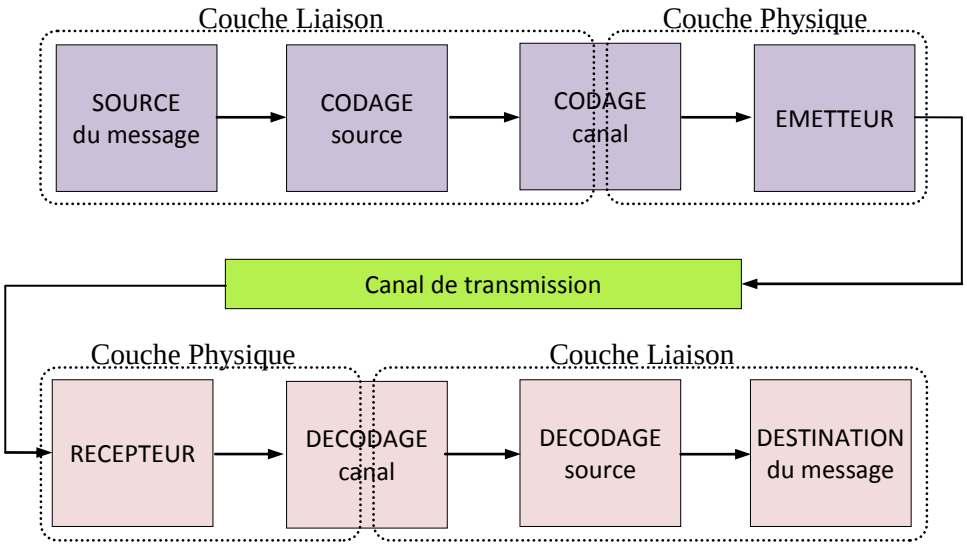


FIG. 1.4 – *Représentation fonctionnelle des couches physique et liaison d'un système de transmission comprenant une paire émetteur-récepteur*

La source du message correspond aux données fournies par la couche liaison de l'émetteur, et réciproquement, la destination du message correspond à la couche liaison 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 aux bruits qui peuvent intervenir lors du transit sur le canal de transmission.

Les données numériques à émettre sont présentes sous la forme de grandeurs abstraites n'ayant de signification logique que pour le système numérique sur lequel

elles sont produites. Il est donc nécessaire de leur associer une représentation physique concrète, qui peut se matérialiser par exemple, sous forme d'un signal électrique. La fonction de l'émetteur correspondante est la modulation, qui associe à chaque groupe de n éléments du message source un signal de durée $T = n \cdot T_s$ où T_s est le période de symbole [2]. Cette association s'effectue au niveau de l'amplitude, de la fréquence, ou bien de la phase du signal transmis. Il est également possible d'utiliser une combinaison de plusieurs de ces paramètres. Le nombre et les genres de modifications qu'il est possible d'apporter au signal de sortie sont définis par le type de modulation choisie. Si les données sont groupées par paquets de n bits, il y aura en tout $M = 2^n$ états possibles. L'émetteur est également chargé d'effectuer une fonction de filtrage et le cas échéant de transposition de fréquence afin d'éliminer les fréquences parasites, et de centrer le signal modulé autour d'une fréquence centrale souhaitée compatible avec le support de transmission [3].

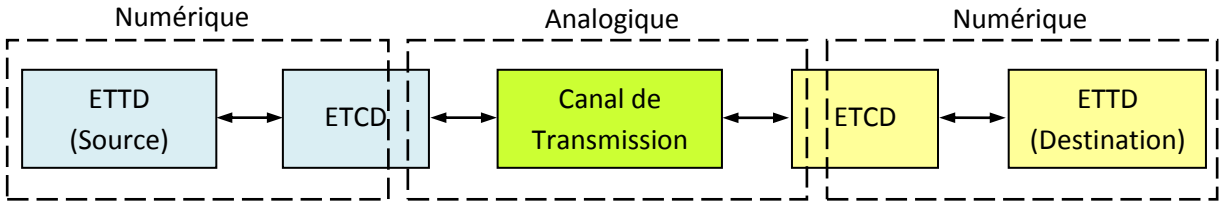


FIG.1.5: *Ligne de transmission, séparation analogique numérique*

La figure 1.5 permet de mettre en valeur la séparation entre les deux types de traitements qui sont effectués sur une ligne de transmission numérique. Le bloc ETTD[1] correspond à l'ensemble des fonctionnalités qui ne sont pas directement liées à la production des signaux à transmettre sur le canal. Le bloc ETCD[2] quant à lui est chargé de la fonction de transformation des signaux M-aire en signaux numérique.

L'ETCD est souvent désigné par le terme de "modem", contraction de modulateur/démodulateur [4]. L'adaptation au support nécessite souvent la modulation

[1] *Équipement Terminal Transmetteur de Données*
[2] *Équipement Terminal de Circuit de Données*

et la démodulation d'une fréquence porteuse. Car le signal produit par la source dans sa forme brute a une occupation spectrale très grande, et le transmettre sans modification, en bande de base, n'est souvent pas compatible avec le canal de diffusion. Il est nécessaire de lui apporter des modifications pour qu'il corresponde aux caractéristiques de ce canal.

L'opération de modulation vise en particulier à transposer la bande de fréquences occupée par ce signal dans une plage adaptée au médium. Pour une liaison numérique bidirectionnelle simultanée ou à l'alternat, il est nécessaire que chaque ETCD prenne en charge les deux fonctions. Les modems disposent d'une interface analogique avec le canal de transmission, et d'une interface numérique avec l'ETTD auquel ils sont associés.

Le passage d'un mode de gestion numérique des données à un mode analogique s'effectue dans l'ETCD et, en ce qui concerne les architectures traditionnelles, en amont de la fonction de modulation. Cette transition est effectuée à l'aide de convertisseurs analogique/numérique.

1.2.3. Structuration fonctionnelle de la couche physique À tous et contraintes d'un traitement numérique

Une partie importante de l'équipement de transmission de données est réalisée à l'aide de circuits ou de composants analogiques [2]. Les architectures qui sont implantées de cette façon atteignent des fréquences de fonctionnement importantes. Mais ce sont surtout les avancées technologiques et physiques qui permettent de repousser plus haut encore ces vitesses. Par contre, les circuits considérés ne se prêtent pas à une évolution des protocoles ou des modulations liées, et il est nécessaire de produire de nouveaux circuits à chaque changement de norme.

Les circuits FPGA, tout comme les ASIC[1], fonctionnent à l'aide de signaux échantillonnés et doivent se conformer aux contraintes de l'échantillonnage que se traduisent en particulier le critère de Nyquist et Shannon, lesquels seront étudiés un peu plus loin dans la section 1.3. Ces contraintes font que les circuits numériques ne sont pas aussi rapides que les circuits analogiques correspondants, et leurs performances brutes s'en ressentent. Par contre, ils disposent d'autres avantages :

– il n'y a pas de dérive des caractéristiques en fonction des conditions extérieures ;
– la précision des informations traitées est garantie par le nombre de bits choisi pour représenter les nombres ;
– il est possible d'implanter directement des fonctions de filtrage avancées qui ne sont pas accessibles dans le domaine analogique ;
– l'intégration avec les étapes de codage ou de préparation des signaux est bien meilleure.

Pour pouvoir traiter des signaux dans une bande passante importante, il est nécessaire de disposer d'une vitesse de calcul élevée [5]. Les circuits de type ASIC bénéficient de technologies de fabrication autorisant des performances relativement élevées. Mais leur structure interne est figée : ils ne possèdent donc pas les mêmes possibilités d'évolution que les circuits FPGA. Ces derniers disposent d'à tous spécifiques :

– les possibilités de prototypage sont infinies, et la simulation logicielle aisée ;
– il est possible de modifier l'architecture de traitement, ou le type de modulation utilisée pour s'adapter à de nouveaux protocoles sans changer de composant ;
– les coûts inhérents au développement d'applications, et ceux des composants sont très inférieurs à ceux des ASIC.

[1] *Application Specific Integrated Circuit*

1.3. Transmission d'un signal numérique en bande transposée

La transmission d'un signal numérique peut être faite en bande de base, en utilisant un simple codage adapté. Cependant, l'encombrement spectral de tels signaux est théoriquement infini, bien qu'il soit possible de le réduire à un intervalle fini à l'aide d'un filtrage adapté [6]. Mais la bande de fréquence occupée par un signal en bande de base dépend directement du type de codage utilisé et du débit de symboles binaires choisi. Ce type de signal est particulièrement sensible aux phénomènes d'atténuation et de distorsion lesquels sont d'autant plus prononcés que la distance entre l'émetteur et le récepteur est grande. La distorsion limite à terme de fréquence de transmission possible en bande de base [4].

Les communications à haut débit utilisent un autre mode de transmission, bien plus efficace, mais qui nécessite d'adapter le signal à l'aide d'une modulation [7, 8]. Cette opération a pour but de produire un signal dont le spectre est transposé pour le rendre conforme à la bande passante du support de transmission. Elle permet aussi le multiplexage fréquentiel permettant à plusieurs systèmes d'accéder simultanément à un même canal [2].

1.3.1. Échantillonnage des signaux

L'échantillonnage est l'opération qui permet de représenter de façon discrète un signal continu dans le temps. Le signal échantillonné correspond aux valeurs du signal original, prélevées à des intervalles de temps fixes, multiples d'une durée donnée et qui est appelée période d'échantillonnage T_e.

- ***Le théorème de l'échantillonnage*** **[9]** présente les conditions dans lesquelles un signal continu peut être complètement représenté par la suite discrete de ses échantillons. En d'autres termes, un signal est complètement représenté si à partir de

ses échantillons prélevés à intervalles T_e, il est possible de reconstruire l'intégralité du signal initial.

Ce théorème s'exprime de la façon suivante : un signal dont le spectre ne contient aucune composante de fréquence supérieure ou égale à une fréquence maximale f_{max} peut être entièrement représenté par la suite des valeurs constituées par ses échantillons, à condition que ceux-ci soient prélevés à des intervalles régulièrement espacés, dont la durée T_e n'excède pas :

$$T_e = \frac{1}{2f_{max}} \tag{1.1}$$

On peut alors définir la fréquence d'échantillonnage par $f_e = \frac{1}{T_e}$.

1.3.2. Cas des signaux sinusoïdaux

L'opération d'échantillonnage présente un intérêt tout particulier en ce qui concerne les signaux sinusoïdaux. Car c'est de cette opération que découlent les architectures d'un grand nombre de générateurs de signaux numériques : en utilisant une quantité limitée d'échantillons stockés dans des mémoires, il est possible de reconstruire un ensemble complet de signaux de fréquence variée [6].

Soit le signal sinusoïdal suivant :

$$s(t) = \cos(\omega t + \varphi) \tag{1.2}$$

L'échantillonnage de s conduit à la production de la suite $s(n)$ telle que $s(n) = \cos(\omega n T_e + \varphi)$, avec $T_e = \frac{1}{f_e}$ la période d'échantillonnage.

1.3.3. Quantification d'un signal

Un signal échantillonné se présente comme une suite d'impulsions dont les amplitudes peuvent prendre une infinité de valeurs distinctes. La quantification consiste à réduire le nombre de valeurs que peut prendre cette amplitude, en attribuant à chaque échantillon une valeur choisie parmi un ensemble fini de valeurs prédéterminées. Ces valeurs sont les niveaux de quantification.

Tous les échantillons ayant une amplitude réelle différente, mais se trouvant dans une même plage d'échelonnage se verront attribuer la même valeur de quantification.

La quantification d'un signal introduit une erreur, qui produit du bruit de haute fréquence lors de la reconstruction du signal analogique à la fin du traitement numérique. Ce bruit de quantification $\varepsilon_q(t)$ correspond à l'équation

$$\varepsilon_q(t) = m(t) - m_q(t) \tag{1.3}$$

où $m(t)$ correspond au signal source, et $m_q(t)$ au signal quantifié.

De plus, certaines techniques de modulation utilisent des approximations de la valeur de quantification des échantillons pour simplifier les calculs sur ceux-ci ou pour réduire la surface occupée par une fonction logique. Ces approximations peuvent en partie être absorbées par le filtre passe-bas en sortie de l'étage numérique. Celui-ci permet, entre autres, d'éliminer une partie des fréquences indésirables. Cependant, plus les erreurs et approximations sont élevées, plus l'ordre du filtre de correction doit être grand.

Ceci peut engendrer un coût important, d'autant plus que dans le domaine des transmissions haut débit, ce filtre doit pouvoir fonctionner avec des signaux de fréquence très élevée.

1.3.4. Production numérique de signaux en bande transposée

La génération de signaux en bande transposée consiste, à partir d'un signal numérique ou analogique donné, à transposer sa fréquence centrale pour que son spectre se retrouve dans une bande de fréquence bien définie. Cette bande de fréquence dépend principalement des caractéristiques physiques du canal de transmission. Les protocoles utilisés et les formes de modulations qu'ils préconisent sont conçus pour se conformer au maximum à ces caractéristiques [6, 5, 10, 11].

La production à l'aide d'architectures numériques de signaux en bande transposée doit se conformer aux règles de l'échantillonnage et de la quantification. Les performances d'un système numérique seront donc dépendantes :

– de la vitesse maximum de traitement des échantillons numériques ;

– de la précision avec laquelle les calculs sur ces échantillons sont faits.

Un système numérique fonctionnant à une vitesse d'horloge élevée permettra donc d'atteindre un débit d'échantillons important. Plus la résolution en nombre de bits des échantillons est importante, plus les opérations arithmétiques qui sont effectuées sur ces échantillons pour produire un signal modulé, ou pour le démoduler, ont une influence néfaste sur la vitesse de fonctionnement du circuit numérique [5].

Pour obtenir les meilleures performances, il est donc primordial d'utiliser autant que possible des opérateurs et des fonctions adaptées spécifiquement à l'architecture numérique cible.

1.4. Architectures analogiques pour la modulation et la démodulation numériques

Dans cette section, nous présentons la structure d'un modulateur QAM analogique et du démodulateur associé. Bien que la modulation QAM ne soit qu'une des méthodes existantes pour transmettre un signal numérique sur une fréquence porteuse, elle illustre, de par la complexité des opérateurs qu'elle met en œuvre, les contraintes que pourra poser à terme son implantation sur une architecture numérique.

1.4.1. Modulateur QAM analogique

Une modulation QAM peut être produite de deux façons distinctes :

– elle peut être considérée comme une modulation d'amplitude et de phase, et combine alors les techniques utilisées pour ces deux types de modulation ;

– il est également possible de moduler en amplitude deux porteuses en quadrature, puis de reconstituer le signal résultant à l'aide d'un additionneur. C'est cette technique qui est la plus fréquemment utilisée [2] et qui est présentée ci-après.

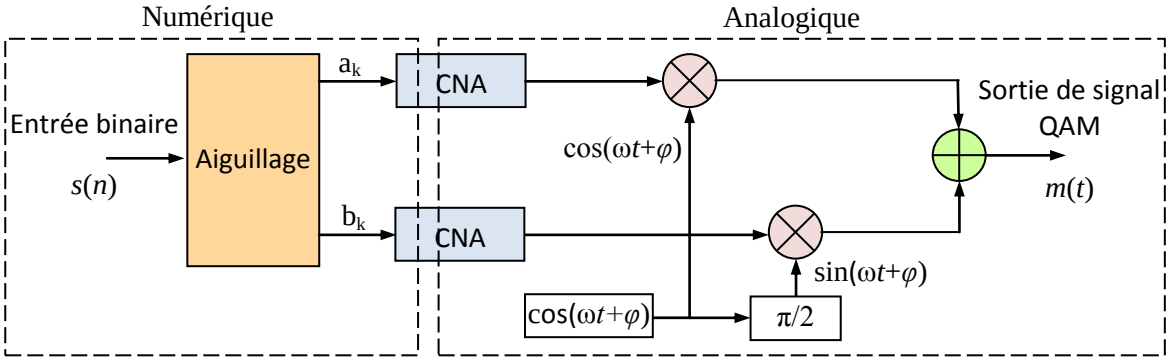


FIG. 1.6: *Architecture d'un modulateur analogique QAM*

Le modulateur correspondant est présenté sur la figure 1.6. Il est notamment constitué d'un bloc d'aiguillage qui associe à chaque série de M bits en provenance de l'émetteur un couple de valeurs correspondant aux amplitudes des composantes sinusoïdales en phase et en quadrature.

Les fonctions de génération des sinusoïdes, représentées par le bloc $\cos(\omega t + \varphi)$ sont le plus souvent implantées à l'aide de boucles à verrouillage de phase. Les termes d'amplitude sont appliqués à l'aide de deux multiplieurs, l'un pour la composante en phase, l'autre pour la composante en quadrature.

Les deux sinusoïdes sont alors combinées à l'aide d'un additionneur pour produire le signal modulé à émettre $m(t)$.

1.4.2. Démodulateur QAM analogique

Le démodulateur QAM correspondant est présenté sur la figure 1.7. Deux chemins de calculs sont utilisés, l'un pour la composante en phase, l'autre pour celle en quadrature. Plus précisément, les deux composantes sont séparées grâce aux multiplications par $\cos(t\omega + \varphi)$ et $\sin(t\omega + \varphi)$.

Les blocs de prise de décision sont ensuite chargés d'effectuer une discrimination parmi l'ensemble discret des différentes valeurs possibles d'amplitude. La valeur résultante est communiquée à un bloc numérique qui reconstitue la suite de bits correspondante.

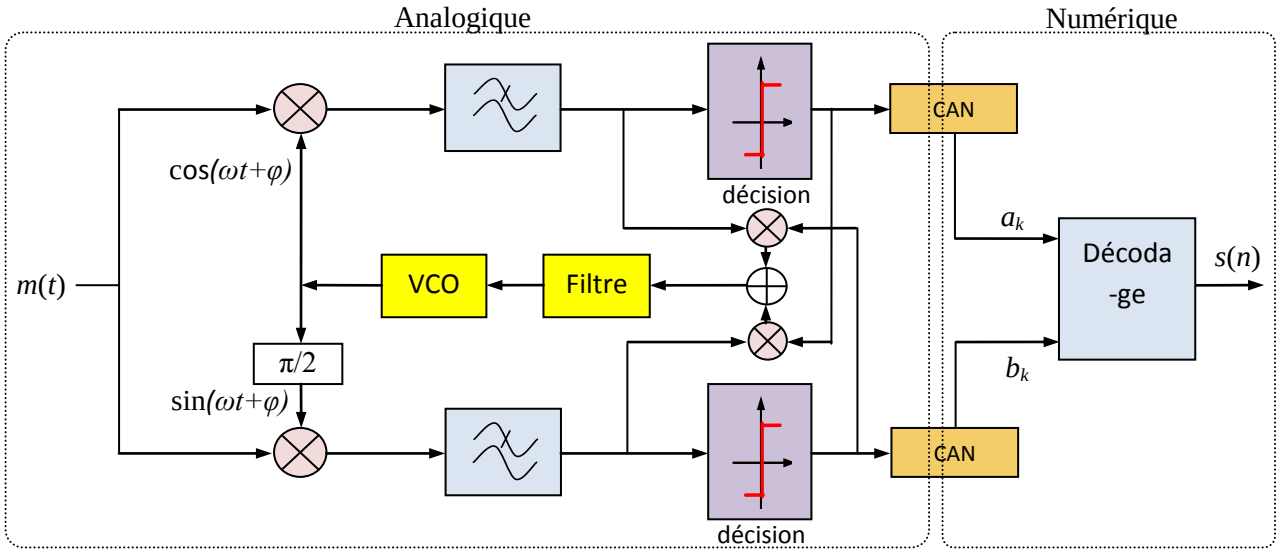


FIG.1.7- *Architecture d'un démodulateur analogique QAM*

La structure du démodulateur est beaucoup plus complexe que celle du modulateur QAM, en particulier parce que le spectre des signaux modulés QAM ne contient aucune raie à la fréquence de l'onde porteuse. La boucle utilisée dans cet exemple pour récupérer la phase de l'onde porteuse, est appelée boucle de Costas. Elle est composée de deux multiplieurs, d'un additionneur, d'un filtre et d'un oscillateur contrôlé en tension (*VCO*[1]). La sinusoïde en phase correspondant à l'onde porteuse est reconstituée par interpolation et corrections successives à partir des valeurs numériques produites par les deux blocs de décision. Deux filtres passe-bas placés en amont de ces blocs de décision permettent de supprimer les composantes à fréquence élevées qui subsistent après la multiplication.

1.4.3. Incompatibilités avec le traitement numérique

Aussi bien dans le démodulateur que dans le modulateur, on trouve des fonctionnalités dont la mise en œuvre sur un circuit numérique limite excessivement la

[1] *Voltage Controlled Oscillator*

vitesse de fonctionnement, tout en occupant une surface ou un nombre de cellules logiques importants. Les difficultés sont posées principalement par le générateur de sinusoïdes, représenté sous la forme d'un bloc $\cos(t\omega + \varphi)$ sur la figure 1.6, et par l'oscillateur contrôlé en tension, noté *VCO* sur la figure 1.7. Dans un circuit numérique utilisant une logique synchrone le chemin le plus lent entre deux registres est appelé «chemin critique». Or, l'ensemble des opérations arithmétiques effectuées dans ces boucles se traduit par un chemin critique très long si elles sont implantées sur une architecture numérique. La fréquence de fonctionnement d'une architecture numérique dépend essentiellement de la taille des chemins critiques [12]. Une architecture numérique utilisant ce type de boucle serait sans doute contrainte de recourir à un composant analogique externe pour permettre des performances acceptables.

De plus, les multiplieurs utilisés pour supprimer les porteuses en quadrature et en phase dans le démodulateur, ou pour produire ces même composantes dans le modulateur, doivent fonctionner à une fréquence élevée pour garantir une précision suffisante au signal de sortie et être compatibles avec des fréquences d'échantillonnage élevées.

1.5. Circuits FPGA dans le cadre de la modulation et la démodulation

Les nouvelles générations de circuits FPGA fournissent sur une même puce l'équivalent de plusieurs centaines de milliers de portes logiques, ainsi qu'un certain nombre de bancs de mémoire vive. Les avantages des FPGA sur les systèmes de type ASIC sont doubles :

– ils proposent tout d'abord un flot de conception plus court ne nécessitant pas d'étape industrielle de production de circuit ;

– ensuite, il est possible de passer directement du prototype au produit final en utilisant dans les deux cas la même plate-forme.

D'un autre côté, les ASICs permettent d'atteindre des performances supérieures, en particulier en ce qui concerne les fréquences de fonctionnement, et les fonctionnalités qu'il est possible d'y implanter. Cependant, les FPGA s'avèrent être une solution de compromis intéressante pour systèmes produits en petit volume.

	ASIS	**FPGA**	**CPU ou DSP**
Performances	Très élevées	élevées	Faibles
Taille	Faible	Moyenne	Elevée
Consommation	Faible	Moyenne	Très élevée
Intégration	Système sur une puce	Système sur une puce	Composants annexes nécessaires
Souplesse	Fonction figées	Reconfigurable	Programmable
Mise en œuvre	Complexe	Moyennement complexe	Faiblement à moyennement complexe
Coût composants	Très élevée	Moyen	Faible

TAB. 1.1- *Comparaison des FPGA par rapport aux autres architectures de traitement du signal*

Le tableau 1.1 présente les avantages et inconvénient respectifs des architectures les plus couramment utilisées pour le traitement numérique du signal. Les circuits FPGA apparaissent comme un bon compromis, tant du point de vue de leur prix de revient limité que de leurs performances leur permettant d'être intégrés dans les architectures de traitement du signal à haut-débit. Les sections suivantes présentent à titre d'exemple l'architecture interne d'un FPGA [13] qui est le circuit cible pour l'ensemble des architectures proposées dans ce document.

1.5.1. Technologie FPGA du Xilinx

Dès l'origine, les FPGA, tels que Xilinx les a inventés, avaient la réputation de mettre à disposition de l'utilisateur une conception rapide, fiable et simple. Les progrès technologiques ont permis d'accéder à des matrices logiques programmables de

plusieurs millions de portes. Cette complexité actuelle reste absolument gérable et permet la réalisation d'applications très performantes moyennant une bonne connaissance des ressources offertes et le respect d'une méthodologie de conception.

La Figure 1.8, montre l'architecture simplifiée d'un FPGA Xilinx (pour les familles Spartan-IIE, Virtex-E et Virtex-II). Les principales caractéristiques de ces trois familles sont :

- Complexités allant de 1500 à plus de 8 millions de portes.
- Faible consommation.
- Grande souplesse d'utilisation des entrées/sorties avec adaptation d'impédance (Virtex-II) et configuration en mode différentiel (Spartan-IIE, Virtex-E et Virtex-II).
- Fonctions mémoire (distribuée et blocs de Ram).
- Dispositifs de gestion des horloges (DLL et DCM).
- Multiplieurs câblés (Virtex-II)

Et bien d'autres possibilités permettant d'optimiser à la fois la performance et la densité des fonctions logiques et/ou arithmétiques.

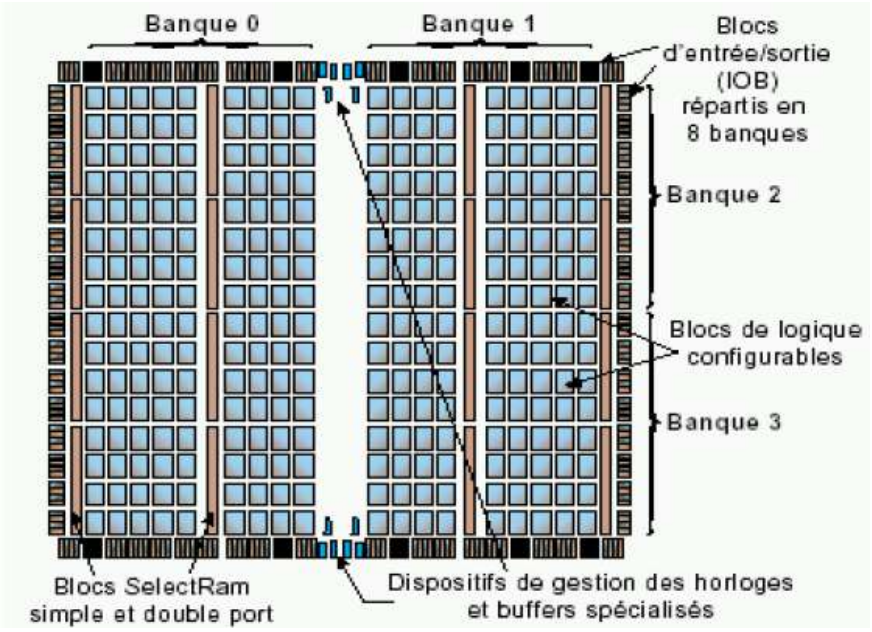


FIG.1.8 − *Organisation des FPGA de Xilinx*

Nous y voyons qu'un des modules de base est le bloc logique configurable (ou CLB[1]). Lui-même constitué de s*lice*. La Figure 1.9, illustre l'architecture simplifiée d'un slice. La logique combinatoire est implantée grâce aux LUT[2] contenues dans chaque slice. Ces LUT peuvent également être configurés comme éléments de mémoire synchrone, simple ou double-port de 16 bits, ou encore comme registres à décalage de 16 bits.

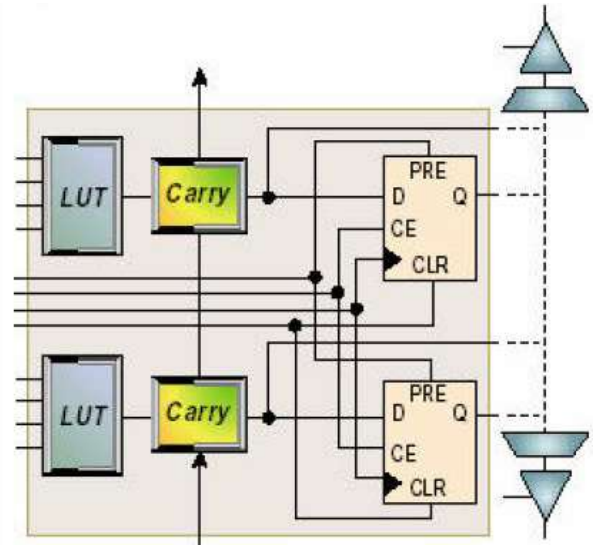


FIG.1.9 − *Architecture simplifiée d'un « Slice »*

Il existe donc trois modes de configuration de ces LUT. Plus précisément, le fonctionnement en mode combinatoire est obtenu en lisant le contenu pointé par les signaux d'entrée (Figure 1.10a).

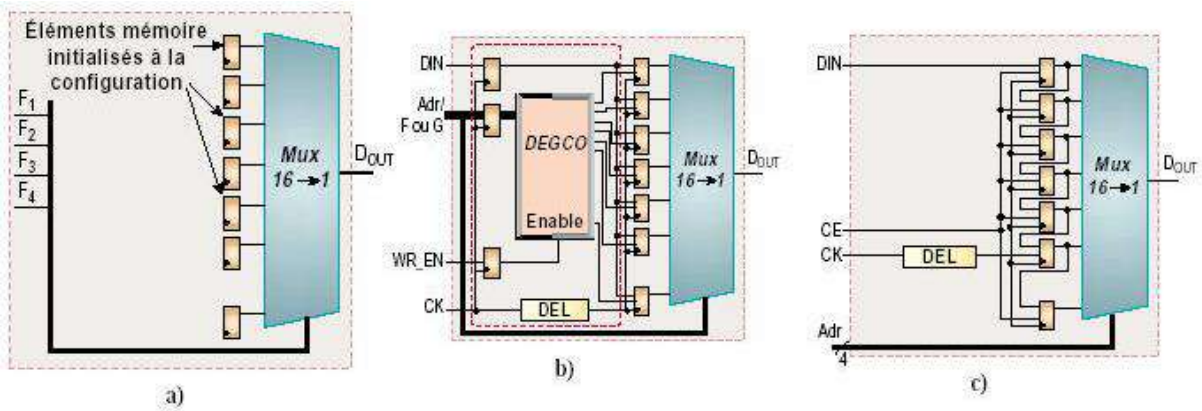


FIG.1.10 − *Trois modes de configuration des LUT*

[1] *Configurable Logic Block*
[2] *Look-Up Table*

Autrement dit, les LUT sont des mémoires dont le contenu est initialisé lors de la configuration du FPGA. De ce fait, elles permettent à l'utilisateur d'en disposer en mode « élément mémoire » dans chacun des slices si nécessaire (Figure 1.10b). La Figure 1.10c décrit le mode de configuration particulier en registre à décalage de longueur programmable jusqu'à 16 bits. Par ailleurs, une logique supplémentaire utile pour la réalisation des fonctions arithmétiques est disponible dans chaque slice. Grâce à ces éléments, et au style d'écriture adapté, des modules de type accumulateur chargeable en addition/soustraction pourront être implantés à raison de 2 bits par slice.

Les bascules dans chaque slice ont aussi des caractéristiques importantes pour le concepteur. En particulier, elles sont initialisées systématiquement à la mise sous tension (par défaut à valeur '0'), et sont utilisables indépendamment de la logique combinatoire disponible dans le même slice. En outre, chaque bascule bénéficie de broches de contrôle telles que : entrée dédiée de validation de l'horloge (*Clock Enable*) permettant d'activer ou de suspendre le fonctionnement de chacune des bascules individuellement, et ceci sans avoir à insérer de la logique combinatoire sur le chemin de l'horloge (entrée de « *set* » et de «*reset* » synchrones ou asynchrones). La polarité des signaux d'horloge, de «*Clock Enable* », de *set* et *reset* est programmable pour chacune des bascules. Autrement dit, ces signaux peuvent être individuellement actifs au niveau haut ou au niveau bas.

L'échange des données avec les circuiteries externes (microprocesseur, DSP[1], mémoires rapides, convertisseurs CNA[2] et CAN[3]) est souvent un critère important dans la perspective des performances à atteindre. Les blocs d'entrées/sorties disposent de bascules sur les chemins d'entrées/sorties et de contrôle à trois-états. Les outils de synthèse les plus sophistiqués permettent l'utilisation systématique de ces bascules.

Enfin, les FPGA de la famille Virtex-II intègrent des registres doubles sur chacun de leurs trois chemins, autorisant ainsi la réalisation d'interfaces DDR[4] pour la

[1] *Digital Signal Processor*
[2] *Convertisseur Analogique Numérique*
[3] *Convertisseur Numérique Analogique*
[4] *Double Data Rate*

communication avec les mémoires SDRAM[1] du même nom ou autres dispositifs de même type (Figure 1.11).

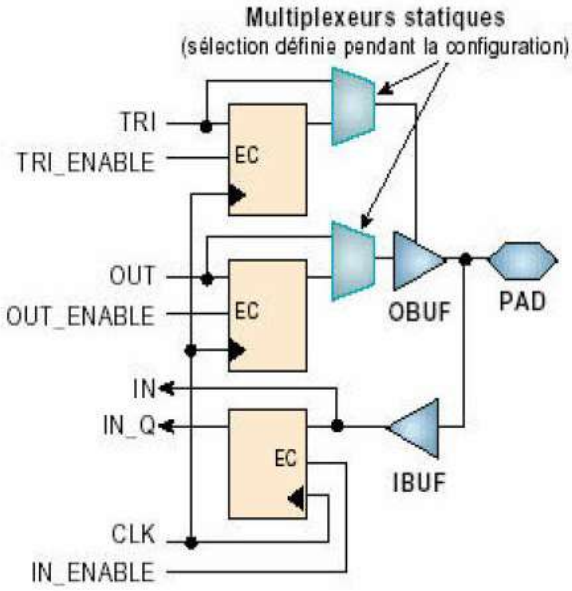


FIG.1.11 − *Configuration des blocs d'E/S*

La configuration électronique des blocs d'E/S donne la possibilité d'ajuster la raideur des fronts sur les étages de sortie, la sortance, les seuils de communication et les standards de communication (LVTTL, LVCmos, SSTL, PCI, LVDS…). Par ailleurs, la famille Virtex-II offre d'adapter en impédance aussi bien les entrées que les sorties, sans avoir besoin d'insérer les traditionnelles résistances d'adaptation.

1.5.2. FPGA Virtex-II

- ***Description général***

La famille Virtex-II est une plate forme développée par FPGA pour haut rendement de basse densité à haute densité des motifs qui sont basés sur des cœurs d'IP[2] et des modules personnalisés. Cette famille propose des solutions complètes pour les

[1] *Static Dynamique Random Access Memory*
[2] *Intellectual Property*

télécommunications, sans fil, applications réseau, vidéo et DSP, y compris Interfaces PCI, LVDS et DDR. L'interconnexion de technologie du 8-couche de pointe 0.15µm /0.12µm CMOS et l'architecture Virtex-II sont optimisés pour haute vitesse avec une faible consommation électrique.

La combinaison d'une grande variété des fonctions souples et une large gamme de densités pouvant atteindre un système de 10 millions portes, la famille Virtex-II améliore les capacités de conception en logique programmable et est une alternative puissante pour dispositif programmable FPGA. Comme montré sur le tableau 1.2, la famille Virtex-II comprend 12 composants, allant à partir de 40K à 8M de system Gates.

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads(1)
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18 Kbit Blocks	Max RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1,536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3,072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7,680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10,752	336	56	56	1,008	8	624
XC2V3000	3M	64 x 56	14,336	448	96	96	1,728	12	720
XC2V4000	4M	80 x 72	23,040	720	120	120	2,160	12	912
XC2V6000	6M	96 x 88	33,792	1,056	144	144	2,592	12	1,104
XC2V8000	8M	112 x 104	46,592	1,456	168	168	3,024	12	1,108

TAB.1.2 – *Ressources du circuit VIRTEX-II*

- ***Architecture de Virtex II***

Les circuits Virtex-II sont des matrices de portes programmables par l'utilisateur avec divers éléments configurables. L'architecture Virtex-II est optimisée pour des conceptions logiques à haute densité et haute performance. Comme le montre la figure

1.12, le dispositif programmable est composé par des blocs d'entrée/sortie (IOBs[1]) et des blocs logiques configurables internes (CLBs[2]).

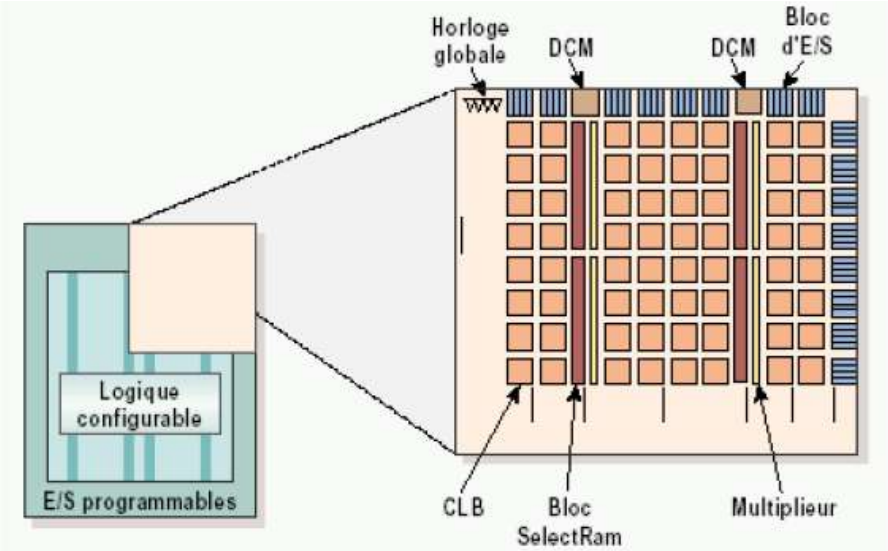


FIG.1.12 − *Détail de l'architecture d'un composant Virtex-II*

Les blocs «SelectRam » dans l'architecture Virtex-II sont des mémoires Ram double port de 18Kbits, configurables de différentes manières entre 18x1Kbit et 512x36 bits. Chaque port est totalement synchrone et indépendant. Les blocs «SelectRAM » peuvent être mis en cascade pour réaliser de larges zones de stockage enfouies dans le FPGA. Un module multiplieur 18x18 bits est disposé à proximité de chaque bloc « SelectRam » et optimisé pour opérer sur le contenu d'un port de la mémoire.

Les fonctions mémoire «SelectRam » et les multiplieurs sont connectés aux matrices d'interconnexions, cette configuration leur ouvrant l'accès aux ressources de routage globales de la matrice FPGA.

[1] *input/output blocks*
[2] *configurable logic blocks*

Dans la famille de Virtex-II, on s'intéresse particulièrement au composant XC2V1000, puisque c'est le composant qui supporte notre projet.

1.5.3. Limitations et avantages pour le traitement numérique du signal

La structure générale d'un FPGA en fait un outil de premier plan pour implanter des algorithmes de traitement du signal et notamment, les algorithmes de modulation et de démodulation qui nous intéressent. Cependant, nous avons vu dans les sections 1.4.1 et 1.4.2 que les architectures analogiques pour la modulation et la démodulation numériques mettent en œuvre des fonctions complexes et en particulier un grand nombre de multiplieurs et/ou de diviseurs.

Un multiplieur peut être implanté en utilisant par exemple le *mode arithmétique* des éléments logiques. Il est par exemple possible de l'implanter en tirant partie des chemins optimisés qui permettent de cascader plusieurs cellules pour réaliser une fonction complexe, et des propagations de retenues rapides. Mais la complexité dans le domaine des circuits FPGA est d'abord mesurée par le nombre d'élément logiques, qui constituent la partie élémentaire du circuit. Ceci est différent dans les circuits de type ASIC, où cette complexité est évaluée en nombre de portes. Une seule cellule logique peut être équivalente à une architecture ASIC comprenant de une à 12 portes. De plus pour les FPGA, le nombre de cellules logiques élémentaires disponible est limité dès la conception du circuit. Lorsque le taux de remplissage du FPGA approche de sa capacité limite, la vitesse de fonctionnement se trouve très affectée par les délais d'interconnexion entre les différents éléments logiques.

Par conséquent, lors de la conception d'une architecture numérique dont la cible est un circuit FPGA, il est nécessaire de limiter la complexité des fonctions arithmétiques et logiques dans les limites imposées par la taille des structures élémentaires disponibles. Autant que possible, les fonctions ne doivent pas s'étendre

sur plusieurs CLB. Les cellules élémentaires de ces circuits sont composées d'une fonction combinatoire simple suivie d'un registre. Cela favorise les architectures tendant à reproduire cette structure, et ce sont celles-ci qui présenteront les meilleures performances sur ce type de circuit [14].

Les fonctions arithmétiques mises en œuvre dans les architectures numériques décrites plus haut sont trop complexes pour permettre une utilisation optimale d'un FPGA. Il est donc nécessaire de définir de nouvelles fonctions équivalentes, mais mieux adaptées à la structure interne de ces circuits.

1.6. Conclusion

Nous avons présenté dans ce chapitre les fondements théoriques qui régissent la transmission numérique des informations. La position et le rôle des modulateurs et des démodulateurs dans une chaîne de transmission ont été explicité, ainsi que les difficultés d'implantation posées aux architectures numériques par la transmission d'un signal numérique en bande transposée. Deux exemples d'architectures de modulation et de démodulation de signaux QAM nous ont permis de mettre en lumière les éléments fonctionnels de ces architectures qui empêchent leur transcription directe sur un circuit numérique.

En effet, les performances atteintes par les systèmes numériques synchrones, et les FPGA en particulier dépendent fortement de la complexité des opérations arithmétiques à réaliser, et de la longueur maximale du chemin critique entre deux registres. L'analyse de la structure interne d'un FPGA nous a permis de comprendre les avantages, mais aussi les contraintes posées par ces circuits. C'est pour cette raison qu'une nouvelle méthodologie de conception est proposée pour la conception de systèmes numériques pour la transmission de signaux à haut débit.

Elle propose de transcrire dans le domaine numérique les fonctionnalités nécessaires à la modulation ou la démodulation, plutôt que de transcrire les architectures analogiques existantes, trop complexes et mal adaptées au numérique.

Chapitre 2

Architectures Numeriques pour la Modulation Numerique

2.1. Introduction

L'utilisation de circuits de type FPGA pour la modulation numérique de signaux présente de nombreux avantages, en particulier celui d'une validation pratiquement immédiate des fonctionnalités, et d'une facilité d'adaptation inégalée en cas de changement de protocole ou de type de modulation. Cependant, l'implantation sur FPGA nous rend également tributaires de la vitesse limitée de ce type de circuit et de la quantité de ressources (sous forme de cellules élémentaires) disponibles. Ces limitations peuvent être compensées en parallélisant de façon importante les algorithmes utilisés, et en utilisant des fonctions arithmétiques simples, qui tiennent compte des spécificités architecturales de ces composants.

Ce chapitre présente donc quelques techniques permettant d'obtenir une modulation numérique de signaux, en nous focalisant sur la modulation QAM qui présente la particularité de nécessiter le contrôle de deux paramètres de la sinusoïde de sortie, à savoir, sa phase et son amplitude. Elle est donc l'une des modulations les plus complexes utilisées de nos jours dans les protocoles de transmission. Mais les solutions proposées ici restent valides pour d'autres types de modulations.

Il est aussi fait mention d'architectures reposant sur des mémoires stockant tout ou une partie seulement des échantillons et qui privilégient la simplicité par rapport à la surface consommée. Une solution permettant d'améliorer la précision de la sinusoïde produite et qui utilise des filtres interpolateurs sera ensuite étudiée. Nous examinerons l'adéquation de ces systèmes avec une architecture reconfigurable, et la facilité avec laquelle il est possible de les adapter à des modifications de protocoles induisant des changements de fréquence ou de type de modulation. Une architecture utilisant en son cœur un algorithme de type CORDIC modifié est ensuite présentée puis analysée dans la section 2.3.

2.2. Architecture d'un modulateur numérique

Parler de modulateur entièrement numérique peut-être considéré comme un abus de langage puisqu'à proprement parler, certains des composants nécessaires à la génération d'un signal modulé ne peuvent être implantés que sur une architecture analogique. Il s'agit du convertisseur numérique analogique final, ainsi que de l'élément de filtrage permettant de supprimer les composantes à haute-fréquence résiduelles provenant de l'échantillonnage et de la numérisation des informations traitées [9, 6]. Cependant, à l'exception de ces deux parties, l'ensemble des autres fonctions élémentaires du modulateur peuvent être traitées à l'aide de composants numériques.

2.2.1. Les modulations numériques

La modulation a pour objectif d'adapter le signal à émettre au canal de transmission. Cette opération consiste à modifier un ou plusieurs paramètres d'une onde porteuse centrée sur la bande de fréquence du canal, cette onde a la forme suivante :

$$S(t) = A.\cos(\omega t + \varphi) \tag{2.1}$$

Les paramètres pouvant être modifiés sont :

– l'amplitude du signal A ;

– la phase φ ;

– la pulsation ω.

L'ensemble de ces paramètres peut servir à coder les informations que nous souhaitons transmettre sur le canal. Puisque nous nous intéressons ici aux procédés de modulations numériques, l'information à transmettre se présente sous la forme d'une suite de bits, dans les procédés de modulation M-aire, l'information à transmettre à l'aide d'un paramètre qui prend M valeurs. Ceci permet d'associer à un état de

modulation un mot de n digits binaires. Le nombre d'états est donc $M = 2^n$. Ces n digits proviennent du découpage en paquets de longueur définie [4].

Les types de modulation binaire simples sont :

– les modulations numériques d'amplitude, ou ASK[1]: $S(t) = A(n).\cos(\omega t + \varphi)$;
– les modulations numériques de phase, ou PSK[2]: $S(t) = A.\cos(\omega t + \varphi(n))$;
– les modulations numériques de fréquence, ou FSK[3]: $S(t) = A.\cos(\omega(n)t + \varphi)$.

Dans chacun de ces exemples, n est un entier (0 ou 1) représentant le $n^{ème}$ symbole envoyé. D'autres types de modulation existent. C'est notamment le cas lorsque l'information est codée par des variations discrètes de l'amplitude sur deux porteuses sinusoïdales en quadrature (*Quadrature Amplitude Modulation QAM*). Ce dernier type de modulation peut également être considéré comme une modulation mixte d'amplitude et de phase. En effet, l'expression du signal modulé est telle que

$$S(t) = A(n).\cos(\omega t + \varphi) - B(n).\sin(\omega t + \varphi) \quad (2.2)$$

$$S(t) = \mathrm{Re}[\big(A(n) + jB(n)\big)\exp^{j(\omega t+\varphi)}] \quad (2.3)$$

$$S(t) = \sqrt{A^2(n) + B^2(n)}.\cos\left(\omega t + \varphi + \arctan\left(\frac{B(n)}{A(n)}\right)\right) \quad (2.4)$$

Nous nous intéresserons plus particulièrement par la suite aux architectures permettant de produire des signaux modulés selon le schéma QAM.

[1] *Amplitude Shift Keying*
[2] *Phase Shift Keying*
[3] *Frequency Shift Keying*

2.2.2. Structure générale d'un modulateur numérique

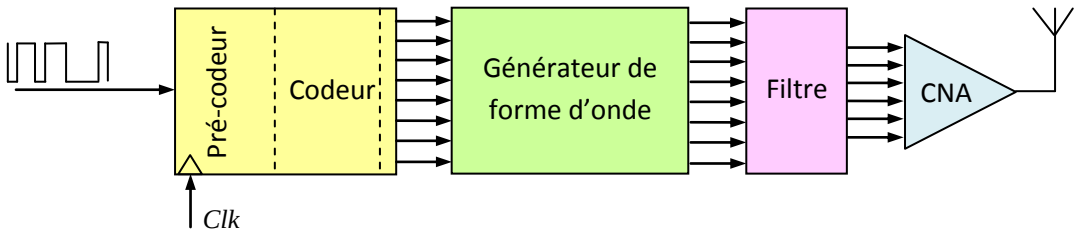


FIG. 2.1 – *Chaîne de modulation numérique QAM*

La chaîne de traitement dans la figure 2.1 indique les tâches réalisées par l'architecture de modulation numérique qui peuvent être composé de quatre parties principales [12], dont les architectures peuvent être étudiées séparément :

– l'accumulateur de phase, dont le rôle est de produire le terme de phase $(\omega t + \varphi)$ dans l'équation 2.1. Ce terme correspond à la phase instantanée de l'échantillon de sinusoïde produit par le modulateur;

– le codeur qui permet de convertir les séries de M-bits composant un symbole, pour les rendre compatibles avec le type de modulation choisi. Pour une modulation d'amplitude par exemple, une série de M-bits sera remplacée par la valeur correspondante de l'amplitude du signal sinusoïdal à produire en sortie du modulateur;

– le générateur numérique de fréquence qui permet de produire les échantillons numériques de sinusoïdes à partir des paramètres que lui fournissent le codeur et, l'accumulateur de phase ;

– le convertisseur numérique analogique, et le filtre passe-bas associé sont les derniers éléments de la chaîne de traitement. Ils convertissent les échantillons en signaux adaptés au médium de transmission qui peut être filaire, hertzien ou

optique par exemple. Il s'agit de la seule partie de l'architecture fonctionnant en mode analogique.

- ***Codeur***

Le codeur est le plus souvent constitué par une mémoire ROM[1] de petite taille, dont la taille d'adressage est égale au nombre de *M* symbole de la modulation considérée. Cette ROM est adressée avec des séquences de *n* bits d'information fournies par les couches supérieures de l'architecture. En sortie de cet étage, on obtient, en fonction du type de modulation l'un ou plusieurs des paramètres suivants :

– F_{cw}, le mot de contrôle de la fréquence ;
– φ_n, le déphasage ;
– A, l'amplitude.

- ***Accumulateur de phase***

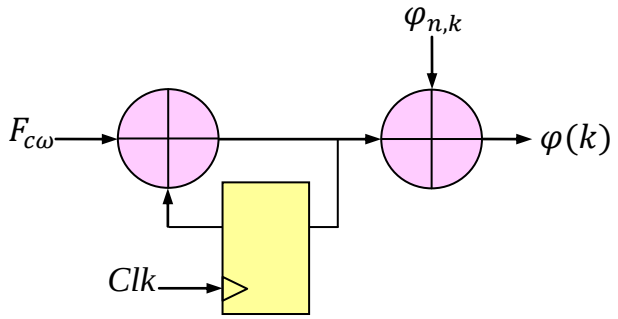


FIG. 2.2 – *Architecture de l'accumulateur de phase*

L'architecture générale de l'accumulateur de phase est présentée sur la figure 2.2. Elle est composée principalement de deux additionneurs et d'un registre. L'accumulateur de phase est piloté par un mot de contrôle de la fréquence F_{cw} correspondant à l'incrément de phase appliqué à chaque période d'horloge du système, N est la longueur (en bits) de l'accumulateur de phase.

[1] *Read Only Memory*

F_{cw} détermine la fréquence f effective résultante du signal modulé, en fonction de la fréquence d'horloge f_{clk} de l'architecture hôte et selon l'équation :

$$f = \frac{f_{clk}.F_{cw}}{2^N} \tag{2.5}$$

L'accumulateur de phase applique également, le cas échéant, un *déphasage* φ_n lorsque la modulation considérée utilise ce paramètre pour transmettre des informations. Le k^e mot de phase produit par cet étage vaut :

$$\varphi(k) = k \cdot F_{cw} + \varphi_{n,k} \tag{2.6}$$

Il faut noter que cette opération est effectuée modulo 2^n, et il est donc garanti que $0 \leq \varphi(k) \leq 2^N$.

- ***Générateur de sinusoïdes***

La sortie de l'accumulateur de phase est reliée à la partie la plus critique de l'architecture de modulation numérique à savoir, le générateur de sinusoïdes, aussi connu sous le nom de générateur de fréquence, qui doit calculer le cosinus du terme de phase produit par l'étage accumulateur. C'est dans cette partie que se trouve en général, le chemin critique de l'architecture de modulation [15], et c'est sur elle que se portera toute notre attention par la suite. À la k^e période d'horloge du système hôte, la valeur

$$S'(k) = \cos(\varphi_k) \tag{2.7}$$

se trouvera en sortie du générateur de sinusoïdes, où φ_k est le mot de phase calculé précédemment par l'accumulateur de phase.

- ***Application du terme d'amplitude***

Suivant le type de modulation considéré, il est parfois nécessaire d'adjoindre aux étapes précédentes un multiplicateur. Celui-ci a pour rôle d'appliquer, le cas échéant

au signal sinusoïdal produit par le générateur de sinusoïdes, le terme d'amplitude issu du codeur, noté A. Cette dernière étape permet d'obtenir un signal final de la forme :

$$S(k) = A\cos\left(k \cdot F_{cw} + \varphi_n\right) \tag{2.8}$$

La chaîne de traitement correspondante se trouve sur la figure 2.1. Notre étude se portera exclusivement sur les blocs pré codage, codage, et générateur de forme d'onde.

2.3. Codage des informations

Dans l'optique de générer une sortie sinusoïdale, il est nécessaire de préparer les informations binaires qui arrivent de manière sérielle en entrée du modulateur. Cette opération est effectuée par le codeur, qui permet à partir du flot de données en entrée d'obtenir une représentation par blocs, compatible avec le générateur de forme d'onde.

Dans notre cas, le but est de supporter un codage adapté à plusieurs types de modulations de phase et/ou d'amplitude. Le codeur acceptera les modulations QPSK et QAM jusqu'à QAM 256.

Afin de simplifier au maximum le circuit, toutes les données, quelque soit le type de modulation choisi, seront représentées dans un format compatible avec la modulation QAM 256. De cette façon il ne sera pas nécessaire d'implanter, dans le générateur de sinusoïdes, différents modules pour l'adapter à chaque type de modulation.

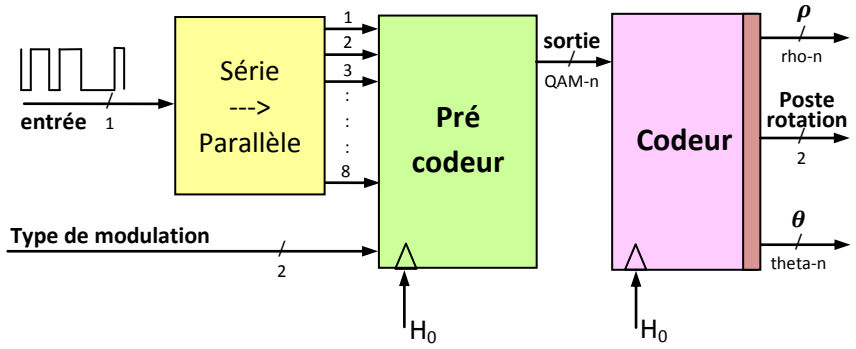


FIG. 2.3 – *Architectures interne de pré-codeur/codeur*

2.3.1. Conception du pré-codeur

Dans la Figure 2.3 le pré-codeur a pour rôle de convertir le signal sériel à son entrée en symboles de type QAM 256 (et donc comportant 8 bits en parallèle) comme indique sur la figure 2.3, parmi ces huit bits sont éventuellement présents des bits nuls de remplissage, qui permettent de compenser le faible nombre de bits par symbole utilisé par les modulations à taille de constellation plus réduite.

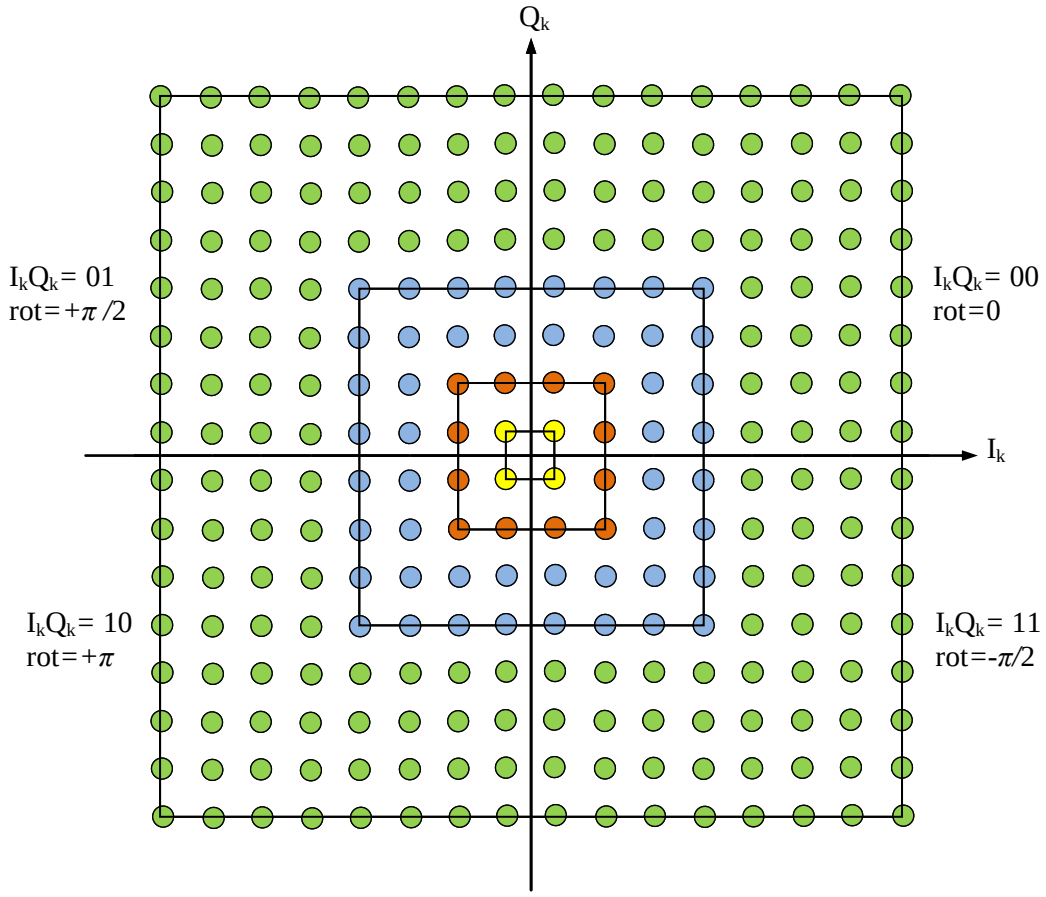


FIG. 2.4 – *Constellations QAM 256, QAM64, QAM16, et QPSK*

La première étape du pré codage consiste en la conversion série / parallèle sur 8 bits du signal en entrée. Puis, en fonction du type de modulation choisi, les données sont converties de manière à pouvoir être placées sur une constellation de type QAM 256. La figure 2.4 montre les propriétés de symétrie de cette constellation.

Du point de vue positionnement des points sur la constellation, il est tout à fait possible d'inclure les constellations QAM 64, QAM 16 et QPSK dans celle de la modulation QAM 256. Par contre, vu quel le nombre de bits par symbole varie suivant le type de modulation, il est nécessaire d'apporter quelques modifications à l'encodage, pour achever l'adaptation. Ces modifications sont représentées sur la figure 2.5.

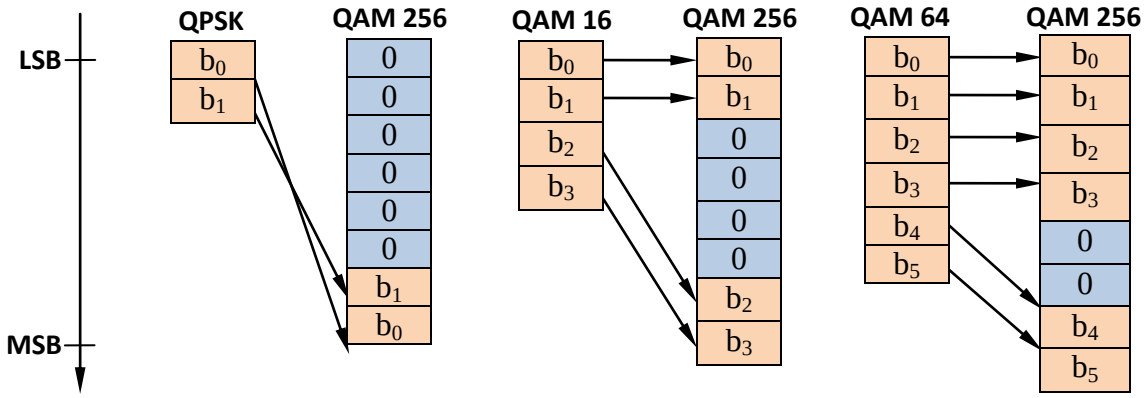


FIG. 2.5 – *Conversion des trames au format QAM 256*

Elles consistent en la permutation de la position de certains bits dans la séquence, ainsi qu'en l'insertion de bits nuls pour compléter la trame QAM 256 à huit bits. Les permutations utilisent les propriétés de symétrie des différentes constellations, puisque la répartition des points dans les différents quadrants de la constellation peut être obtenue simplement par une rotation d'un angle multiple de $\pi/2$ du quadrant principal.

Ainsi, les deux bits de poids fort de chaque symbole représentent le quadrant dans lequel il va se positionner :

– quadrant 1 ($\alpha = 0$) : bits de poids fort 00 ;

– quadrant 2 ($\alpha = +\pi/2$) : bits de poids fort 01 ;

– quadrant 3 ($\alpha = +\pi$) : bits de poids fort 11 ;

– quadrant 4 ($\alpha = -\pi/2$) : bits de poids fort 10.

Une exception est QPSK, pour laquelle les codages des quadrants 2 et 3 sont inversés. En sortie du pré-codeur, nous obtenons donc un bus véhiculant des trames QAM 256, indépendamment du type de modulation souhaitée en entrée.

2.3.2. Conception du codeur

Le codeur qui se situe donc en aval du pré-codeur (Figure 2.3) a également un rôle d'adaptation de signal, à destination du générateur de sinusoïdes. Ici, contrairement à la méthode qui est généralement admise en modulation analogique, qui consiste à générer séparément les deux composantes quadratiques du signal modulé, puis de recomposer celui-ci par une multiplication en bout de chaîne, nous allons plutôt produire les composantes polaires de la sinusoïde, à savoir son amplitude ρ et sa phase θ. La fréquence sera fournie par l'utilisateur à l'aide d'un signal externe au niveau du générateur.

Le codeur devra donc faire correspondre au signal sur huit bits à son entrée, deux signaux en sortie, l'un pour l'angle, et l'autre pour l'amplitude.

- ***Génération de θ, phase de la sinusoïde***

Pour simplifier la génération de l'angle, et diminuer la surface nécessaire pour stocker les valeurs possibles, nous allons utiliser les propriétés de symétrie de la constellation QAM256 vue plus haut, ce qui va également être utile au moment de la génération de sinusoïde, puisqu'il sera possible de restreindre le problème à celui d'une rotation limitée au premier quadrant, à laquelle on adjoint une rotation de correction d'un angle multiple de $\pi/2$. La taille de la table de correspondance trame QAM / angle - amplitude, est donc limitée au quart de ce qu'elle aurait été sans utilisation de la symétrie.

Puisque l'information sur le quadrant est véhiculée par les deux bits de poids fort de la trame QAM 256, seuls les six autres bits seront utiles en fin de compte pour déterminer la valeur de l'angle correspondant à la phase.

- ***Génération de ρ, amplitude de la sinusoïde***

La génération des amplitudes va immanquablement générer des erreurs du fait de l'approximation du codage sur un nombre limité de bits. Pour ne pas à avoir à calculer à la main l'ensemble des amplitudes possibles pour chaque point de la constellation QAM 256, on peut utiliser l'un des logiciel, qui génère automatiquement une matrice contenant les valeurs nécessaires.

2.4. Générateurs de sinusoïdes : architectures existantes

Plusieurs publications et produits industriels s'efforcent de résoudre le problème posé par les architectures numériques permettant de générer des signaux modulés. En particulier pour les modulations complexes, comme la QAM qui contrôle à la fois la phase et l'amplitude du signal considéré, il faut faire face à plusieurs difficultés afin de produire un échantillon M-aire de signal modulé. Le premier problème se trouve dans le calcul du cosinus de la phase de l'échantillon considéré, le second dans la multiplication par le coefficient d'amplitude.

De nombreux travaux existent concernant la synthèse numérique de sinusoïdes. Ce problème est souvent présenté dans les publications comme la conversion d'une valeur de phase en l'amplitude d'une sinusoïde, et plusieurs architectures ont été proposées pour le résoudre. Les principales utilisent des ROM, dans lesquelles sont stockés les échantillons pré-calculés ou une décomposition en série de Taylor de la fonction cosinus pour la traiter à l'aide de fonctions plus simples [16, 17, 18], voire encore des méthodes hybrides utilisant à la fois des mémoires et des méthodes d'approximation polynomiales [19]. Un schéma de l'architecture basique d'un générateur à base de ROM est représenté sur la figure 2.6.

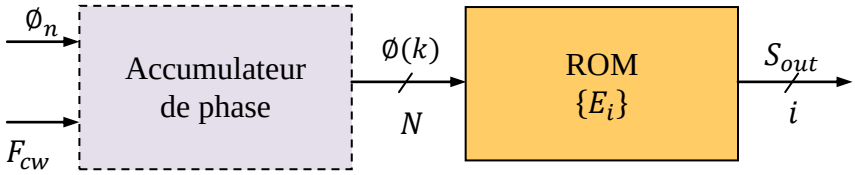


FIG. 2.6 – *Architecture générale d'un générateur de sinusoïdes à base de ROM*

2.4.1. Utilisation de mémoires et d'échantillons pré-calculés

Le type le plus couramment utilisé dans les circuits industriels est basé sur des mémoires en lecture seule qui contiennent les valeurs d'une sinusoïde pré-calculées lors de la conception du circuit. Dans le cas théorique idéal, en accédant à l'adresse *i* de cette ROM, on obtient la valeur

$$E_i = \cos\left(i.\frac{2\pi}{2^n}\right) \tag{2.9}$$

Où $i \in \{0, 2^N - 1\}$. Cette valeur échantillonnée est directement présentée au convertisseur numérique / analogique en sortie, puis au filtre passe bas qui ôte les composantes haute-fréquence résiduelles.

Dans le domaine numérique, la précision des échantillons stockés en ROM est forcément limitée, aussi les valeurs des E_i calculées précédemment sont limitées à l bits en pratique. Le nombre de bits de la ROM résultante est donc de $2^N \cdot l$. Ces conventions étant posées, le k^{e} échantillon disponible à la sortie numérique du générateur de sinusoïdes à base de ROM sera donc de la forme :

$$S_{out}(k) = \cos\left(k.\frac{2\pi}{2^N}\right) \tag{2.10}$$

Le nombre d'échantillons stockés dans la ROM permettra de contrôler l'erreur affectant la phase du signal. En effet, plus il y a d'échantillons calculés, plus l'intervalle de phase entre deux échantillons successifs est faible. De même, la valeur de *l* contrôle la précision en amplitude du signal généré. Plus *l* est grand, plus l'erreur de quantification commise est faible. Pour obtenir une bonne précision, à la fois en phase et en amplitude, il est souhaitable d'utiliser une ROM aussi importante que possible. Cela a cependant des conséquences néfastes sur la fréquence d'horloge maximale que peut atteindre le circuit cible et sur la surface occupée.

- ***Compression de ROM***

Il est possible en pratique de réduire la surface occupée par les ROM chargées de produire les échantillons de sinusoïde. La méthode la plus directe consiste à ne précalculer d'échantillons que pour un seul cadrant du cercle trigonométrique, et à exploiter les symétries de la fonction cosinus pour évaluer les angles situés dans les autres quadrants, grâce à des opérations combinatoires simples [20].

La représentation binaire signée classique des nombres est en complément à deux, ce qui signifie que le bit de poids fort représente notamment le signe. Le second bit, conjointement avec le premier, nous donne une indication sur le sens de variation de l'amplitude. Ces deux bits permettent de détecter le quadrant dans lequel se trouve l'angle dont il faut calculer le cosinus. Les angles se trouvant dans le second et le quatrième quadrant passent par un étage de complémentation à deux leur permettant d'adresser correctement la ROM qui contient les échantillons du premier quadrant. Une seconde étape de complémentation à 2 est effectuée en sortie de la ROM pour les angles se trouvant dans les quadrants trois et quatre, afin de rétablir la valeur correcte du signe.

Ces deux techniques couplées permettent de réduire la taille de la ROM d'un facteur 4, c'est-à-dire de limiter la taille théorique à $2^{N-2} \cdot l$ bits.

Il est encore possible de réduire la taille requise de la ROM utilisée en plaçant dans la ROM non plus directement les valeurs des échantillons de sinusoïde correspondant aux angles du premier quadrant, mais les éléments de la fonction différence suivante [20] :

$$\Delta(k) = \sin\left(\frac{\pi}{2^{N-1}} \cdot k\right) - \frac{\pi}{2^{N-1}} \cdot k. \tag{2.11}$$

En considérant de plus l'égalité

$$\cos x = \sin\left(\frac{\pi}{2} - x\right), \tag{2.12}$$

ce choix permet de réduire de deux bits la taille des échantillons stockés dans la ROM, car

$$\max\left(sin\left(\frac{\pi}{2^{N-1}} \cdot k\right) - \frac{\pi}{2^{N-1}} \cdot k\right) \approx 0.212 \max\left(\sin\left(\frac{\pi}{2^{N-1}} \cdot k\right)\right) \tag{2.13}$$

Il est par contre nécessaire de rajouter un additionneur supplémentaire en sortie de l'étage de calcul. Il est possible de paralléliser le calcul effectué par l'additionneur en, utilisant par exemple une structure en pipeline, ce qui autorise des gains substantiels en termes de vitesse de fonctionnement. *A contrario*, il est extrêmement complexe de transformer la ROM utilisée en une architecture pipeline présentant un chemin critique court.

2.4.2. Utilisation couplée de filtres interpolateurs

En contrepartie de concessions sur la qualité spectrale du signal généré, il est possible de réduire encore la taille de la mémoire utilisée pour produire les échantillons de sinusoïde.

- ***Approximation polynomiale***

[41] Afin de réduire plus la taille des ROM utilisées pour le calcul de la valeur de la sinusoïde, il est possible de ne placer dans les ROM qu'une partie des échantillons réellement utiles, puis d'interpoler les valeurs des échantillons intermédiaires à l'aide de polynômes approchant le comportement de la fonction cosinus pour des valeurs d'angle comprises dans un intervalle restreint [17].

Ainsi, les fonctions sinus et cosinus sont décrites respectivement par les polynômes suivants

$$\begin{aligned} \mathcal{A}(n) &= a_k \cdot \mathcal{C}(n)^K + a_{k-1} \cdot \mathcal{C}(n)^{K-1} + \cdots + a_0 \\ \mathcal{B}(n) &= b_k \cdot \mathcal{C}(n)^K + b_{k-1} \cdot \mathcal{C}(n)^{K-1} + \cdots + b_0 \end{aligned} \tag{2.14}$$

Cette approximation introduit une erreur sur la valeur de l'amplitude de la sinusoïde calculée. Cette erreur peut être réduite en augmentant le degré du polynôme mais ceci rend également l'architecture plus complexe. Les travaux présentés dans [17] concluent qu'un polynôme de degrés 2 ou 3 permet d'atteindre des performances satisfaisantes, pour les applications où l'erreur résultante ne doit pas être supérieure à 60 dBc.

- ***Décomposition en intervalles multiples***

Le principe de cette technique consiste à considérer l'intervalle angulaire $[0, 2\pi]$ comme la réunion d'un ensemble de sous-intervalles de dimension fixe ou variable [19, 18].

La fonction sinus est alors approchée sur chacun de ces intervalles, par une expression affine du type

$$s_i(x) = m_i \cdot \left(x - \frac{i}{8}\right) + y_i, \quad i \in [0,7]. \tag{2.15}$$

Les coefficients m_i et y_i sont évalués à l'aide d'un algorithme de génération, afin de minimiser les erreurs et le bruit résultant sur les échantillons de sinusoïde produits par ce système.

Il n'est pas possible, sur les circuits FPGA disponibles actuellement, d'atteindre les fréquences d'échantillonnage nécessaires à la génération des sinusoïdes à haute-fréquence utilisées pour les transmissions haut-débit. Aussi, il est nécessaire d'utiliser des architectures fortement parallèles afin de dupliquer le nombre d'échantillons produits à chaque cycle d'horloge.

Les architectures que nous venons de décrire sont conçues pour ne produire dans le meilleur des cas qu'un seul échantillon de sinusoïde par cycle d'horloge. Ceci n'est pas suffisant pour répondre aux besoins des transmissions haut débit, car celles-ci nécessitent des signaux de fréquence bien supérieure à celles qu'il est possible d'atteindre sur des circuits FPGA. Il est donc nécessaire de développer des architectures adaptées à une parallélisation ultérieure permettant d'atteindre des débits suffisamment élevés.

2.5. Processeur CORDIC modifié

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 [21, 22] permet de calculer facilement les valeurs des échantillons d'une sinusoïde sans utiliser de multiplication ni de boucle de rétroaction complexe.

CORDIC correspond à l'origine à un algorithme permettant le calcul par approximations successives de fonctions trigonométriques ou bien hyperboliques. Il a été développé par Jack E. Volder dans [21], mais a bénéficié de nombreuses améliorations et extensions par la suite. L'intérêt de cet algorithme est qu'il permet l'évaluation trouvé sur les architectures des microcontrôleurs simples ou sur les FPGA pour lesquels, nous l'avons vu, l'implantation d'un multiplieur, même avec des cellules dédiées, reste coûteuse en ressources et limite grandement la fréquence de fonctionnement du circuit cible. On trouvera dans [23, 24, 25] de nombreux exemples de fonctions analytiques pouvant être approchées grâce à CORDIC, mais nous nous limiterons ici aux fonctions *sinus* et *cosinus* qui ont une application directe dans la génération d'un signal modulé.

2.5.1. Principe du processeur CORDIC

Le principe de l'algorithme CORDIC comme décrit par Volder, est relativement simple.

- ***Fondements mathématiques***

Considérons un vecteur $\vec{v}$ de coordonnées $\begin{pmatrix} x \\ y \end{pmatrix}$. On souhaite lui appliquer une rotation d'angle θ.

Le vecteur $\overrightarrow{v'}$ résultant aura pour coordonnées $\begin{pmatrix} x' \\ y' \end{pmatrix}$ telles que :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ y\cos\theta + x\sin\theta \end{pmatrix} \tag{2.16}$$

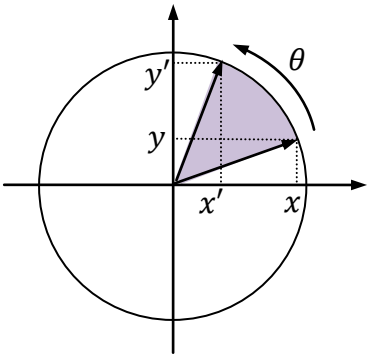


FIG. 2.7 – *La rotation θ sur un cercle trigonométrique.*

Il est alors possible de mettre le terme $\cos\theta$ en facteur dans cette expression, ce qui nous donne :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \cos\theta \begin{pmatrix} x - y\tan\theta \\ y + x\tan\theta \end{pmatrix} \tag{2.17}$$

Donc pour la rotation matricielle d'angle θ du vecteur $\vec{v}\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ sur le cercle trigonométrique, s'écrire sous la forme suivante :

$$\overrightarrow{v'} = \cos\theta \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix} \cdot \vec{v}. \tag{2.18}$$

L'idée réside dans le fait de ne considérer ensuite que les rotations d'angles tels que

$$\theta = \arctan 2^i, \quad i \in \mathbb{Z}. \tag{2.19}$$

L'équation peut alors être simplifiée, et devient

$$\overrightarrow{v'} = \cos(\arctan 2^i) \cdot \begin{pmatrix} 1 & -2^i \\ 2^i & 1 \end{pmatrix} \cdot \vec{v}. \tag{2.20}$$

Une fois implantée sur un circuit fonctionnant en logique binaire, ou sous forme de code machine pour un microcontrôleur, l'opération de rotation correspond à un simple décalage binaire à droite ou à gauche, en fonction du signe de i. Il faut encore cependant multiplier le vecteur résultant par une constante valant $\cos(\arctan 2^i)$. Par

conséquent, ce type de calcul est particulièrement bien adapté pour les systèmes qui nous intéressent, puisqu'il substitue à une opération trigonométrique complexe, une multiplication par une puissance de deux, soit un simple décalage en binaire.

La situation décrite ci-dessus ne considère qu'un sous ensemble particulier d'angles. Il est cependant possible d'étendre la méthode quelle que soit la valeur de l'angle considéré.

- ***Généralisation***

En effet, pour une valeur de θ prise dans l'intervalle $]-\frac{\pi}{2},\frac{\pi}{2}[$, Volder a démontré qu'il existe une suite de valeurs d_i, avec $d_i \in \{-1,1\}$, pour laquelle

$$\theta = \sum_{i=0}^{\infty} d_i \cdot \arctan 2^{-i} \,. \tag{2.21}$$

En termes géométriques, présentés sur la figure 2.8, la rotation d'angle θ peut être décomposée en une série de micro-rotations successives qui obéissent aux critères de l'équation 2.21, et pour lesquelles seul le *sens* de la rotation effectuée dépend de θ.

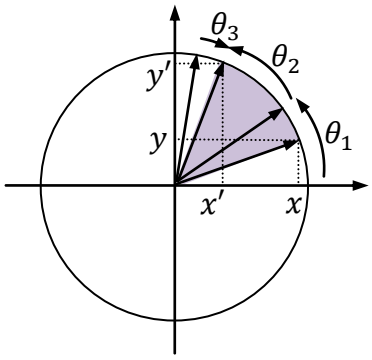


FIG. 2.8 – *Décomposition en plusieurs micro-rotations.*

Ainsi, effectuer une rotation d'angle θ peut s'écrire sous la forme matricielle suivante:

$$\overrightarrow{v'} = \prod_{i=0}^{\infty} \cos(d_i \cdot \arctan 2^{-i}) \cdot \prod_{i=0}^{\infty} \begin{pmatrix} 1 & -d_i \cdot 2^{-i} \\ d_i \cdot 2^{-i} & 1 \end{pmatrix} \cdot \vec{v}. \qquad (2.22)$$

Pour simplifier cette expression, on peut poser :

$$\begin{aligned} C_\infty &= \prod_{i=0}^{\infty} \cos(d_i \cdot \arctan 2^{-i}) \\ &= \prod_{i=0}^{\infty} \cos(\arctan 2^{-i}), \text{ car la fonction cos est paire} \\ &\approx 0{,}6073. \end{aligned}$$

En d'autres termes, si $\vec{v}_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix}$ est le vecteur obtenu à la k^e itération des produits décrits dans l'équation ci-dessus, la micro-rotation permettant de passer de $\vec{v}_k$ à $\vec{v}_{k+1}$ est équivalente au système :

$$\begin{cases} x_{k+1} = x_k - d_k \cdot y_k \cdot 2^{-i} \\ y_{k+1} = y_k + d_k \cdot x_k \cdot 2^{-i} \end{cases} . \qquad (2.23)$$

Il est donc possible d'effectuer de façon itérative la rotation complète d'angle, en n'utilisant que des opérations simples d'addition et de décalage à gauche ou à droite. Cette suite récurrente converge vers la matrice solution du problème initial, à savoir

$$\overrightarrow{v'} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \vec{v}. \qquad (2.24)$$

2.5.2. Application à la génération paramétrique de sinusoïdes

Le principe de la génération de sinusoïdes consiste à produire en sortie de notre architecture une onde de la forme $S(t) = A \cdot \cos(\omega t + \varphi(t))$. L'expression

échantillonnée de ce signal est donc $S(n) = A \cdot \cos(\omega T_e \cdot n + \varphi(n))$, si $\frac{1}{T_e}$ est la fréquence d'échantillonnage considérée. Il est donc nécessaire de calculer pour chacun des échantillons, la valeur du cosinus correspondant. L'algorithme 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. En choisissant un vecteur $\vec{v}$, et un angle θ tels que

$$\vec{v} = \begin{pmatrix} A \\ 0 \end{pmatrix} \tag{2.25}$$

et

$$\theta = 2\pi \cdot \omega t + \varphi(t) \tag{2.26}$$

l'algorithme produit une rotation qui conduit au vecteur résultant

$$\vec{v'} = \begin{pmatrix} A\cos\big(2\pi \cdot \omega t + \varphi(t)\big) \\ A\sin\big(2\pi \cdot \omega t + \varphi(t)\big) \end{pmatrix}. \tag{2.27}$$

Cet algorithme récursif présente cependant trois types de limitations dans le cadre de l'implantation sous forme de circuit numérique.

- ***Approximation de la fonction trigonométrique***

La première est qu'il ne fournit un résultat exact qu'au bout d'un nombre *infini* de rotations successives. Cet aspect doit être pris en considération, mais n'a que peu d'effet sur les architectures numériques qui nous intéressent. Car la précision des opérations arithmétiques effectuées est dans tous les cas bridée par la résolution binaire du microprocesseur ou de la logique implantée. Il suffit donc d'effectuer suffisamment de micro-rotations pour que l'erreur de calcul effective ne dépasse pas la résolution du circuit cible. Si on arrête le calcul au bout de *N* étapes, soit *N* micro-rotations effectuées, l'erreur commise sur l'angle de la rotation réellement effectuée est telle que

$$\sum_{i=N}^{\infty} \arctan(2^{-i}) \leq \varepsilon_N, \tag{2.28}$$

et pour *N* suffisamment grand, $arctan(2^{-i}) \approx 2^{-i}$. Ce qui conduit à

$$2^{-N} \underbrace{\sum_{i=0}^{\infty} 2^{-i}}_{\lim_{\to\infty}=2} \leq \varepsilon_N \tag{2.29}$$

$$2^{-N+1} \leq \varepsilon_N \,. \tag{2.30}$$

La condition nécessaire et suffisante pour obtenir un résultat à une précision donnée de n_b bits est donc que $N \geq n_b + 1$.

- ***Calcul du sens des micro-rotations***

La seconde limitation de cet algorithme est de nécessiter la connaissance *a priori* du sens de chacune des micro-rotations successives. La suite $\{d_i\}_{i\in\mathbb{N}}$ correspondant à chacun des angles pour lesquels le cosinus doit être évalué peut être stockée dans une mémoire. Cependant, cette solution n'est pas envisageable dans le cadre d'une implantation numérique sur ASIC ou FPGA, de par l'occupation de surface trop importante. Mais il est possible de prévoir directement en ligne le sens de la prochaine micro-rotation à effectuer, en rajoutant au système une équation dont le rôle est de conserver en permanence *l'angle de la rotation qu'il est encore nécessaire d'effectuer pour aboutir à une rotation d'angle* θ [26]. Après chaque micro-rotation, cette angle est augmenté – ou diminué suivant le sens de cette micro-rotation – de la valeur 2^{-i}. Si l'angle restant est supérieur à 0, alors la prochaine rotation se fera dans le sens anti trigonométrique. Dans le cas contraire, elle se fera dans le sens trigonométrique. Ces considérations se traduisent par une modification des équations qui deviennent :

$$\begin{cases} x_{k+1} = x_k - d_k \cdot y_k \cdot 2^{-i} \\ y_{k+1} = y_k + d_k \cdot x_k \cdot 2^{-i} \\ z_{k+1} = z_k - d_k \cdot \arctan 2^{-i} \\ d_{i+1} = -1 \text{ si } z_{k+1} \geq 0, +1 \text{ sinon.} \end{cases} \tag{2.31}$$

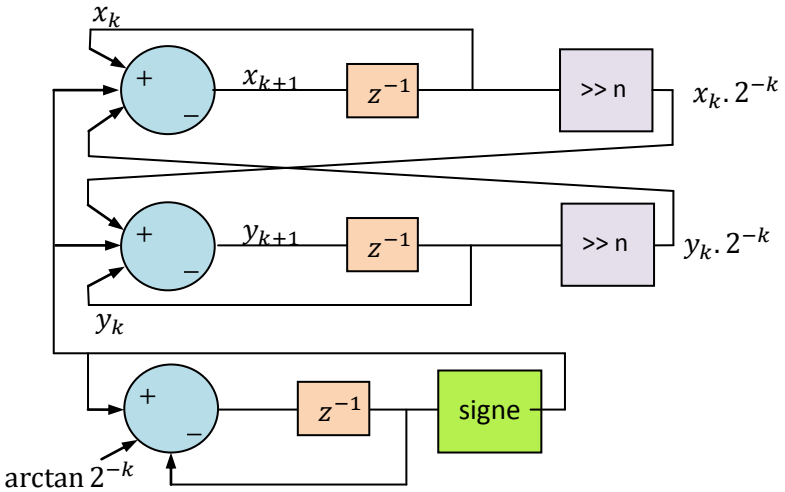


FIG. 2.9 – *Cellule élémentaire d'un processeur CORDIC*

La figure 2.9 présente la cellule e base du processeur CORDIC que nous venons de décrire. On y retrouve les trois chemins de calcul *X*, *Y* et *Z*. La complexité arithmétique se trouve dans les additionneurs soustracteurs, et dans les décaleurs, comme nous le verrons plus tard.

- ***Facteur d'échelle***

Enfin, la dernière limitation provient du terme C_∞ qui correspond à une erreur effectuée sur la norme du vecteur $\overrightarrow{v'}$ résultant. Mais cette valeur, appelée facteur d'échelle, ne dépend que du nombre de micro-rotations effectuées, et non pas de leur sens. Il est donc possible de la compenser *à priori*, en choisissant un vecteur $\overrightarrow{v}$ tel que

$$\overrightarrow{v} = \begin{pmatrix} C_\infty \cdot A \\ 0 \end{pmatrix}. \tag{2.32}$$

On remarquera que dans ce cas qu'il est possible de prendre en compte directement le terme d'amplitude du signal produit en sortie, et cela *sans nécessiter de multiplieur supplémentaire* à la sortie de l'étage de génération de sinusoïdes.

2.5.3. Architecture CORDIC itérative

L'algorithme présenté plus haut peut être implanté *en l'état* sur un circuit numérique. C'est d'ailleurs bien souvent sous cette forme qu'il est utilisé sur les microcontrôleurs pour permettre le calcul de fonctions trigonométriques, voir hyperboliques.

L'architecture proposée pour le générateur numérique de sinusoïdes est présentée sur la figure 2.10. Il est composé de trois éléments principaux :

– une ROM, contenant les valeurs pré-calculées des $arctan2^{-i}$;

– un bloc combinatoire, chargé de calculer les valeurs successives de x_i et y_i ;

– un bloc combinatoire, chargé d'évaluer l'angle restant.

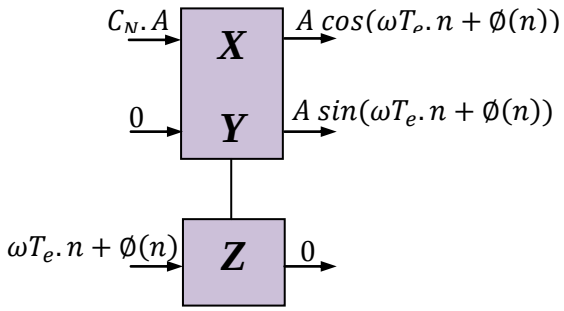


FIG. 2.10 – *Générateur de sinusoïdes à base de processeur CORDIC itératif.*

Au début du fonctionnement du circuit, les blocs X et Y sont pré-chargés avec les valeurs $C_N \cdot A$ et 0 respectivement, et Z est chargé avec la valeur de l'angle dont il faut calculer le cosinus, à savoir... . À chaque itération, les nouvelles valeurs de X et Y sont calculées en fonction des valeurs de l'étape précédente, et du sens de la rotation donné par le signe de Z. La valeur de Z est modifiée pour refléter la rotation qu'il reste à effectuer pour atteindre la rotation d'angle θ. Le calcul est terminé lorsque $Z = 0$, à la précision près du circuit sur lequel cette architecture est implantée. Il faut environ

$n + 1$ itérations pour obtenir la valeur d'un échantillon de sinusoïde avec une précision de n bits.

La position des registres est telle qu'à chaque cycle d'horloge, X Y et Z sont mis à jour une fois. Par conséquent, $n + 1$ cycles d'horloges seront nécessaires pour produire un échantillon. La fréquence d'échantillonnage théorique correspondante sera donc de $\frac{f}{n+1}$, où f est la fréquence de fonctionnement du circuit sur lequel est implantée cette architecture. De plus, cette architecture utilise :

– trois additionneurs soustracteurs complets sur n bits ;
– une ROM comportant $n + 1$ valeurs $arctan 2^{-i}$, chacune ayant une précision de n bits, soit $n^2 + 1$ bits au total ;
– deux décaleurs à barillet sur n bits, dont la complexité est comparable à celle d'un additionneur de même taille.

L'architecture présentée ici a l'avantage d'être très proche de l'algorithme CORDIC, et d'occuper relativement peu de place [27, 28]. Mais elle fait appel à des décaleurs à barillet qui engendrent un bloc combinatoire volumineux et dont le chemin critique est important. De plus, nous cherchons à obtenir la plus grande fréquence d'échantillonnage possible, et cette architecture nécessite plusieurs cycles d'horloge pour produire un seul échantillon, ce qui n'est généralement pas satisfaisant.

2.5.4. Architecture CORDIC pipeline

La principale limitation de l'architecture précédente est qu'elle restreint fortement la fréquence d'échantillonnage théoriquement possible, d'un facteur au moins égal à la précision en bits des échantillons de sortie. Il est cependant possible, mais au prix d'une occupation de surface plus élevée, de produire un échantillon de sinusoïde par cycle d'horloge, en utilisant une version pipeline de l'algorithme CORDIC [29].

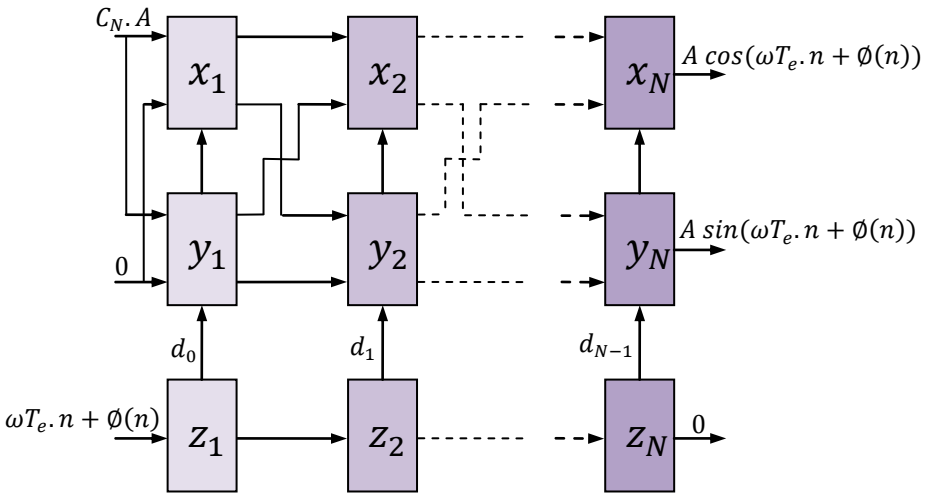


FIG. 2.11 – *Générateur de sinusoïdes à base de processeur CORDIC pipeline.*

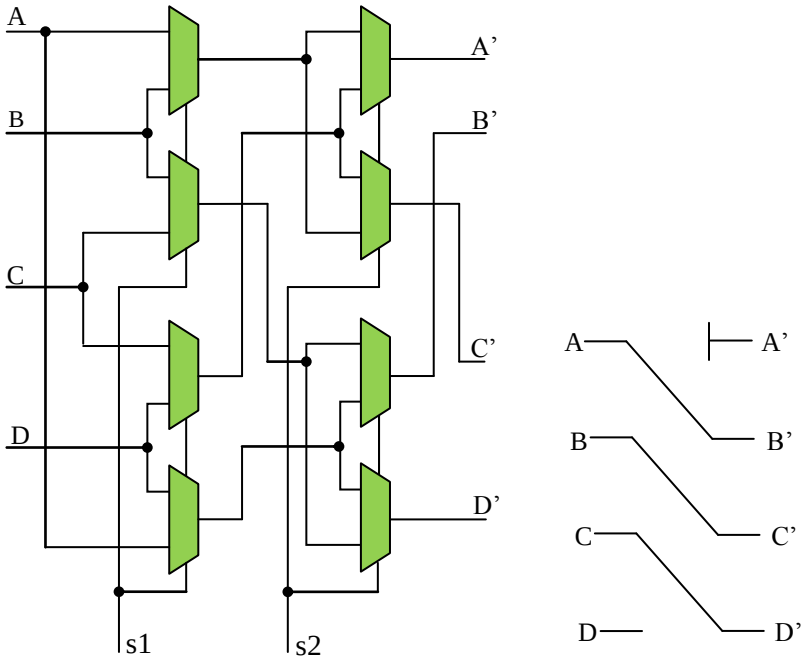
L'architecture modifiée est présentée sur la figure 2.11. Elle comporte un pipeline dont la profondeur ne dépend que de la précision souhaitée pour les échantillons de sinusoïde. Il y a $n+1$ étages sur le schéma présenté, pour une précision théorique de n bits. Chaque étage du pipeline n'est chargé d'effectuer qu'une seule micro-rotation, toujours la même, dont seul le sens peut être modifié. C'est-à-dire que le k^e étage correspond au système d'équations :

$$\begin{cases} d_k = signe(z_k) \\ x_{k+1} = x_k - d_k \cdot y_k \cdot 2^{-i} \\ y_{k+1} = y_k + d_k \cdot x_k \cdot 2^{-i} \\ z_{k+1} = z_k - d_k \cdot \arctan 2^{-i} \end{cases} . \tag{2.33}$$

Cette fois, la fin du calcul n'est plus déterminée par le fait que le contenu de Z vaut 0, mais par le fait que $n+1$ micro-rotations ont été effectuées. D'après les résultats présentés dans la section 2.3, cela est suffisant pour obtenir un échantillon de sinusoïde à la précision binaire désirée.

En plus du fait de produire un échantillon à chaque cycle d'horloge, cette architecture possède l'avantage de ne pas utiliser de décaleurs à barillet puisque pour

chaque étage du pipeline, le décalage à effectuer est connu par avance et ne dépend que de k.



(a) Décaleur à barillet (b) Décaleur à câblé

FIG. 2.12 –*Deux architectures de décaleurs*

Comme le montre la figure 2.12, ce décalage peut être implanté par câblage sans nécessiter d'élément logique supplémentaire. Le chemin critique de cette architecture n'est plus affecté par cet élément mais uniquement par les additionneurs nécessaires au calcul des valeurs de X, Y et Z. Par conséquent, ce chemin n'est pas plus long que celui d'un additionneur complet sur N bits. En résumé, l'architecture pipeline utilise :

– $N + 1$ additionneurs soustracteurs complets pour chacun des trois chemins de calcul (X, Y et Z) ;

– les valeurs des $arctan2^{-i}$ sont implantées sous forme de constantes, et ne consomment donc pas de surface ;

– $2(N + 1)$ décalages, ne nécessitant pas d'élément de logique combinatoire.

La fréquence théorique de production d'échantillons de sinusoïde est donc égale à la fréquence d'horloge du circuit sur lequel est implantée cette architecture.
La version pipeline de l'algorithme CORDIC présente un avantage indéniable sur la version itérative, si le gain de vitesse est notre priorité [28]. Mais la version pipeline occupe une surface importante qu'il est encore possible de réduire en supprimant l'accumulateur d'angles Z.

2.5.5. Approximation de la valeur des $arctan2^{-i}$

Afin de pouvoir supprimer l'accumulateur d'angles, il est nécessaire de trouver une représentation simplifiée des $\arctan 2^{-i}$ stockés jusqu'à maintenant sous forme de constantes. On peut pour cela constater que, pour une valeur *x* suffisamment petite

$$\arctan x \approx x. \tag{2.34}$$

Pour un système numérique dont la résolution est finie et déterminée par le nombre de bits *n* utilisés pour exprimer les valeurs numériques, ceci est équivalent à

$$\arctan x = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} x^{2i+1} \tag{2.35}$$

$$= x - \frac{x^3}{3} + \frac{x^5}{5} \cdots \tag{2.36}$$

Ce développement est valable pour les valeurs de *x* telles que $|x| < 1$. On peut déduire de cela l'inégalité

$$|\arctan x - x| < \frac{|x^3|}{3}\ , \tag{2.37}$$

pour $x \neq 0$. Cette condition, appliquée à l'équation 2.37, nous permet d'affirmer que l'approximation est valable pour les valeurs de *x* telles que

$$\frac{|x^3|}{3} < 2^{-n+1}. \tag{2.38}$$

En appliquant ce résultat aux angles particuliers utilisés dans les architectures CORDIC qui sont de la forme $x = 2^{-k}$, l'équation précédente devient

$$\frac{|2^k|^3}{3} < 2^{-n+1}. \tag{2.39}$$

Cette expression peut être simplifiée de la façon suivante

$$\ln\left(\frac{2^{-3k}}{3}\right) < \ln(2^{-n+1})\frac{n-1}{3} - \frac{\ln 3}{3\ln 2} < k. \tag{2.40}$$

Par conséquent, l'équation

$$k > \frac{n-1}{3} \tag{2.41}$$

est une condition suffisante pour que l'approximation $arctan2^{-k} = 2^{-k}$ soit acceptable à la précision binaire du circuit utilisé.

- ***Extension de l'approximation***

On souhaite utiliser des rotations liées directement aux bits de l'angle, mais sans pour autant introduire de multiplications. Pour cela, on cherche à utiliser l'approximation $\arctan 2^{-k} = 2^{-k}$ pour décider du sens de rotation à effectuer dans le processeur CORDIC, quelle que soit la valeur de k. Pour $k \leq \frac{n-1}{3}$, une erreur est donc introduite dans la micro-rotation correspondante. Afin de s'affranchir de cette erreur, il est possible d'exprimer, pour $k \leq n$, la valeur de 2^{-k} en fonction des $\arctan 2^{-i}$, pour $i \leq n$. Par exemple, pour $n = 8$:

$$\begin{cases} 2^{-1} = \arctan 2^{-1} + \arctan 2^{-5} + \arctan 2^{-8} \\ 2^{-2} = \arctan 2^{-2} + \arctan 2^{-5} + \arctan 2^{-8} \\ 2^{-3} = \arctan 2^{-3} \\ \quad\vdots \\ \quad\vdots \\ 2^{-8} = \arctan 2^{-8} \end{cases} \tag{2.42}$$

Ainsi, pour effectuer une rotation d'angle 2^{-2} au lieu d'une rotation d'angle $\arctan 2^{-2}$, il faut effectuer en tout trois rotations intermédiaires. La première d'angle $\arctan 2^{-2}$, la seconde d'angle $\arctan 2^{-5}$, et la dernière d'angle 2^{-8}. À partir de $k \geq 3$, il n'est plus nécessaire d'effectuer de rotations de correction.

Mais chacune des micro-rotations rajoutées modifie le facteur d'échelle d'une quantité égale à $\sqrt{1+2^{-2i}}$ pour une rotation d'ordre i. Il est nécessaire d'étudier si cela a une influence sur le facteur d'échelle global.

- ***Influence sur le facteur d'échelle***

La contribution au facteur d'échelle global des rotations de rang supérieur ou égal à k est négligeable à la précision binaire n si :

$$B_k = \prod_{i=k}^{n} \sqrt{1+2^{-2i}} \leq 1 + 2^{-2n} \tag{2.43}$$

En utilisant la même méthode que dans le cas de l'approximation $\arctan 2^{-k} = 2^{-k}$, on montre que l'équation 2.43 est équivalente à :

$$k \geq \frac{n-1}{2} \tag{2.44}$$

Toutes les micro-rotations de l'exemple présenté plus haut répondent à ce critère, elles n'ont donc aucune n'influence sur le facteur d'échelle global.

Ces approximations, et les quelques micro-rotations supplémentaires qu'elles impliquent, vont nous permettre de supprimer l'accumulateur d'angle utilisé dans l'algorithme CORDIC au prix d'un pipeline un peu plus long.

2.6. Utilisation de la structure à double branche du pipeline

Nous avons vu que le processeur CORDIC, avec ou sans accumulateur utilise 2 branches X et Y pour le calcul des fonctions trigonométriques. Pour une utilisation purement mathématique, l'une seulement des branches est nécessaire. Par contre, puisque nous cherchons à produire une sinusoïde, nous voulons obtenir, pour chaque symbole à émettre présenté à l'entrée du générateur, plusieurs échantillons.

Le théorème de Shannon nous contraint à générer au minimum deux échantillons par période du signal de sortie pour des signaux à phase constante1. Pour chaque symbole, nous avons donc à calculer, entre autres :

$$\rho\cos(\omega_0 t_0 + \theta) \text{ et } \rho\cos(\omega_0 t_0 + \theta + \pi) \tag{2.45}$$

où t_0 prend l'un des valeurs suivant : $\frac{T_e}{2}$, $\frac{T_e}{4}$ et $\frac{3T_e}{4}$.

Il est également possible de calculer aisément deux autres échantillons :

$$\rho\cos(\omega_0 t_0 + \theta + \frac{\pi}{2}) \text{ et } \rho\cos(\omega_0 t_0 + \theta + \frac{3\pi}{2}) \tag{2.46}$$

Ces quatre échantillons peuvent être générés pratiquement sans aucun coût supplémentaire à partir des éléments déjà existants. En effet, on peut constater que :

- $\rho\cos\left(\omega_0 \frac{\pi}{\omega_0} + \theta\right) = \rho\cos(\theta + \pi) = -\rho\cos\theta$
- $\rho\cos\left(\omega_0 \frac{\pi}{2\omega_0} + \theta\right) = \rho\cos\left(\theta + \frac{\pi}{2}\right) = \rho\sin\theta$
- et $\rho\cos\left(\omega_0 \frac{3\pi}{2\omega_0} + \theta\right) = \rho\cos\left(\theta + \frac{3\pi}{2}\right) = -\rho\sin\theta$

Or, $\rho \cos\theta$ et $\rho \sin\theta$ sont déjà connus, puisqu'il s'agit des sorties respectives des deux branches du processeur CORDIC, et leurs opposés peuvent être obtenus par complément à deux directement à la sortie du pipeline.

Cette génération peut être avantageusement associée à la rotation de compensation nécessaire en fin de chaîne, pour pallier au fait que l'on ne considère que le quadrant 1 pour de l'expression des angles au niveau du bloc codeur/précodeur. Cette association s'implante avec un simple multiplexeur, commandé par le signal "quadrant", qui permet de choisir l'échantillon correspondant au cadrant qui nous intéresse.

Suivant la nature du filtre en sortie du processeur CORDIC, il peut être nécessaire soit de présenter les échantillons les quatre à la fois soit de les émettre séquentiellement, auquel cas il sera nécessaire de rajouter des registres pour retarder certaines des lignes de calcul par rapport aux autres. La production d'un plus grand nombre d'échantillons par période est également possible, par groupe de quatre, en alimentant l'additionneur en entrée du processeur avec le décalage de phase $\Delta\theta$ devant séparer deux échantillons successifs.

2.7. Conclusion

Nous avons présenté dans cette section une architecture de génération de sinusoïdes utilisant un processeur CORDIC modifié pour répondre aux exigences d'une implantation optimale sur un circuit numérique de type FPGA. L'architecture résultante autorise l'utilisation efficace d'un pipeline, qui permet d'obtenir un chemin critique court et donc une fréquence de fonctionnement du circuit FPGA plus importante. Enfin, la simplification de l'algorithme par la suppression de l'étage accumulateur d'angle, permet de réduire la surface occupée.

Chapitre 3

Architecture Numerique pour la Demodulation Numerique

3.1. Introduction

Tout comme pour les modulateurs, les architectures numériques apportent des avantages certains aux démodulateurs de signaux notamment dans le cas de l'implantation sur circuits de type FPGA. Les éléments fonctionnels présents dans un démodulateur sont du même ordre de complexité que ceux qui composent un modulateur et on retrouve, en particulier dans les architectures les plus couramment utilisées, des boucles à verrouillage de phase et des générateurs programmables de sinusoïdes.

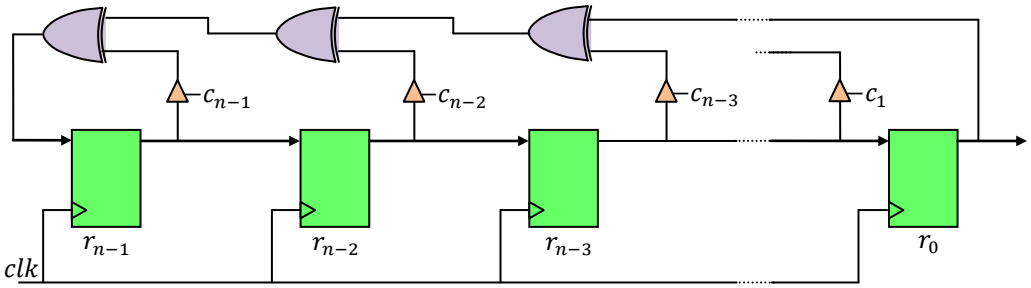
Dans l'optique de s'affranchir des algorithmes complexes se trouvant dans les architectures de démodulation numérique, les registres à décalage à rétroaction linéaire, ou LFSR[1], sont introduits. Leurs atouts pour le traitement de signaux sur architecture FPGA sont particulièrement importants. Leur utilisation en tant que compteurs rapides est étudiée et une nouvelle méthode pour générer facilement un pair compteur/décompteur associés est proposée.

Ces registres sont ensuite utilisés au cœur de nouvelle architecture, pour la démodulation de phase ou de fréquence. Ce système est particulièrement bien adapté aux circuits numériques synchrones. Il atteint en particulier de très bonnes performances sur les circuits FPGA.

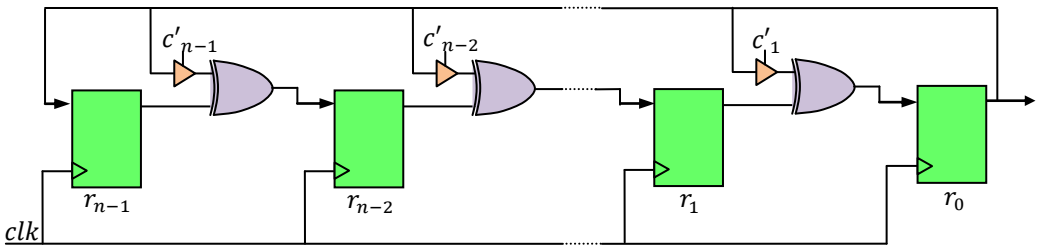
[1] *Linear Feedback Shift Register*

3.2. Registres à décalage à rétroaction linéaire

Dans cette section sont présentés les principes et les fondements mathématiques d'une structure couramment utilisée dans les architectures permettant le traitement de codes correcteurs d'erreur [30, 31], ou plus généralement pour la production de séquences pseudo-aléatoires [32, 33]. Un registre à décalage à rétroaction linéaire – ou LFSR est un type de registre dont l'état des entrées ne dépend que d'une combinaison linéaire de l'état de ses cellules à l'instant précédent.



(a) « many-to-one»



(b) « one-to-many»

FIG. 3.1 – *Les deux architectures de LFSR parmi d'autres.*

Un registre est contrôlé par une horloge qui lui est externe. Dans notre cas, il s'agira de l'horloge système du circuit sur lequel cette architecture est implantée. Ces registres ont des propriétés particulièrement intéressantes, qui permettent entre autre d'envisager leur utilisation en tant que compteurs programmables. Les deux types d'architectures permettant de réaliser des LFSR sont présentés sur la figure 3.1. Un

choix judicieux des coefficients $\{c_i\}_{1\leq i<n}$ et $\{c'_i\}_{1\leq i<n}$ permet d'obtenir de ces deux architectures un comportement équivalent, mais l'architecture de la figure 3.1b est la plus adaptée à l'implantation sur circuit numérique [34]. Les détails concernant ce dernier point sont présentés dans la section 3.2.4.

Les LFSR peuvent être utilisés pour des opérations de démodulation, en tirant parti de leurs propriétés de périodicité qui sont examinées dans les sections suivantes.

3.2.1. Principe mathématique

L'ensemble des principes présentés dans cette section utilisent l'architecture *many-to-one* de la figure 3.1a car celle-ci permet une présentation plus simple des équations. Les propriétés énoncées dans ce paragraphe peuvent également être appliquées à la forme *one-to-many* en utilisant les relations d'équivalence [34].

Le registre est initialisé avec la séquence $R_0 = (r_0, r_1, \ldots, r_{n-1})_0$. À chaque itération, le contenu des cellules est décalé vers la droite, et celui de la cellule la plus à gauche r_{n-1}, est remplacé par une combinaison linéaire des contenus de certaines des cellules du registre.

En conservant les notations présentées sur la figure 3.1a, il est possible d'écrire l'état R_k des cellules d'un LFSR après k itérations en utilisant le système d'équations récurrentes suivant :

$$\begin{cases} r_{k+1,i} = r_{k,i+1}, \text{pour } 0 \leq i \leq n-2 \\ r_{k+1,n-1} = \sum_{j=0}^{n-1} c_j \cdot r_{k,j} \text{ , avec } c_j = 0 \end{cases} \quad (3.1)$$

Ce système peut également être représenté sous la forme matricielle équivalente :

$$R_{k+1} = T^{\oplus} \cdot R_k \quad (3.2)$$

$$\begin{bmatrix} r_{k+1,0} \\ r_{k+1,1} \\ \vdots \\ r_{k+1,n-2} \\ r_{k+1,n-1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \\ 1 & c_1 & \dots & c_{n-2} & c_{n-1} \end{bmatrix} . \begin{bmatrix} r_{k,0} \\ r_{k,1} \\ \vdots \\ r_{k,n-2} \\ r_{k,n-1} \end{bmatrix} \quad (3.3)$$

Avec cette notation les k itérations effectuées sur le vecteur R_0 contenant les valeurs initiales du LFSR se traduisent par l'opération matricielle :

$$R_k = T^k \cdot R_0 \quad (3.4)$$

3.2.2. Périodicité, polynôme minimal et polynôme primitif

Les vecteurs R_i, dans le cas d'une implantation sur circuit numérique, prennent leurs valeurs dans le corps de Galois $\mathbb{GF}(2^n)$, où *n* est la longueur du registre utilisé. Il s'agit d'un corps *fini* et les opérations récurrentes sur cet ensemble sont périodiques [35].

Plus particulièrement l'opération réalisée par le LFSR est périodique, et il existe par conséquent une valeur $m \in N$ telle que $R_m = R_0$, où m définit la période du LFSR.

Il est possible de garantir que cette période soit maximale [32, 30, 36].À cet effet, on définit un polynôme $P(X)$ sur ce corps, appelé *polynôme caractéristique* d'un LFSR de la façon suivante :

$$P(X) = X^n + \sum_{i=0}^{n-1} c_i \cdot X^i. \quad (3.5)$$

Les caractéristiques de la séquence $\{R_i\}_{i\geq 0}$ dépendent de celles de $P(X)$. En particulier, la période de cette séquence est maximale dans le cas où P est un polynôme primitif sur $\mathbb{GF}(2)$ [37]. Cela signifie que P est de degré minimal (en

l'occurrence n) et qu'il est irréductible (il n'a aucun autre diviseur que l'unité et lui-même) [38]. La période maximale d'une telle séquence vaut alors :

$$m_{max} = 2^{n-1} \quad . \tag{3.6}$$

Le plus petit polynôme caractéristique, et donc le plus petit LFSR, permettant de générer la suite de $\{R_i\}_{0 \le i < m}$ peu être calculé si au moins n éléments r_i consécutifs sont connus. Cette opération utilise l'algorithme de Berlekamp-Massey [31, 39] présenté dans la section suivante.

Dans le cas d'un LFSR associé à un polynôme générateur minimal et primitif, le registre parcours, pendant l'ensemble des itérations, une et une seule fois chacun des éléments de l'espace vectoriel $\mathbb{GF}(2^n)$ excepté l'élément nul. L'ordre de parcours ne dépend que du polynôme *P*.

3.2.3. Algorithme de Berlekamp-Massey

Afin de générer des registres LFSR capables de produire une séquence de bits déterminée, Berlekamp [31] a mis au point un algorithme itératif permettant de décoder les codes BCH[1]. Massey, dans [39], a proposé une amélioration de cet algorithme dans le contexte plus spécifique des codes linéaires et a démontré qu'il pouvait être utilisé pour trouver le plus petit LFSR engendrant une suite linéaire donnée $\{r_i\}_{0 \le i < n}$. Des améliorations de ces algorithmes peuvent être trouvées dans [40].

À l'itération d'ordre j cet algorithme produit le polynôme caractéristique du LFSR minimal permettant de générer les j premiers éléments de la séquence $\{R_{i,j}\}$. Il utilise pour cela le polynôme déjà calculé permettant de générer les $j - 1$ premiers éléments.

[1] *Bose, Ray-Chaudhuri, Hocquenghem*

Cet algorithme produit à la demande un LFSR dont la sortie binaire correspond au mieux à l'application dans laquelle il est utilisé. Il est plus particulièrement utilisé dans le logiciel développé pour la description automatique de nos architectures de démodulation, afin de constituer des LFSR de périodicité donnée.

3.2.4. Architectures matérielles pour l'implantation de LFSR

Nous avons présenté sur la figure 3.1a l'une des architectures utilisées pour l'implantation matérielle de registres LFSR. Cette architecture est dite *externe* ou *many-to-one* [34] puisque le fan-in de la fonction combinatoire de rétroaction peut être de 1 à n entrées en fonction de la complexité du polynôme caractéristique.

Ceci pose des problèmes dans le cas d'une optimisation architecturale, car les performances d'un tel système sont alors dépendantes du polynôme générateur $P(X)$. Sur le FPGA utilisé pour nos implantations, chaque cellule logique élémentaire est structurée sous forme d'un bloc combinatoire, comportant trois ou quatre entrées associé à un registre [36]. Les polynômes complexes risquent d'occuper plusieurs cellules et les délais de routage entre ces cellules limitera la fréquence de fonctionnement de l'architecture globale, et à plus forte raison les performances du système résultant.

Aussi, il est possible d'implanter la même fonction séquentielle que celle réalisée par le LFSR de la figure 3.1a en utilisant une architecture pour laquelle la complexité logique de la fonction combinatoire de rétroaction se trouve répartie de façon équitable entre toutes les cellules du registre. Cette architecture est dite *interne* ou *one-to-many* et est également présentée dans [34] et sur la figure 3.1b.

Le polynôme représentatif de l'état des cellules d'un registre de longueur n, après i itérations, devient

$$R_i^{'}(X) = \sum_{j=0}^{n-1} r^{'}_i \cdot X^j. \tag{3.7}$$

Une transformation polynomiale simple permet de passer d'une représentation à l'autre, de façon bijective [34], en écrivant

$$\begin{cases} r'_j = c_{j+1} \cdot r_0 + r_{j+1}, si\ j < n-1 \\ r'_{n-1} = r_0, sinon \end{cases} \tag{3.8}$$

C'est ce type d'implantation qui sera exclusivement choisi pour la suite puisqu'il est mieux adapté aux circuits FPGA.

3.2.5. Compteurs à base de LFSR

Les LFSR, de par leur structure à base de registres et de fonctions logiques simples, sont particulièrement bien adaptés à une implantation sur FPGA [36], car leur architecture est proche de la structure interne des cellules logiques des FPGA présentés dans 1.4 et [13]. De plus, le fait qu'ils soient capables de produire des séquences de longueur fixes et périodiques, comme cela a été démontré dans la section 3.2.2, on fait un choix pertinent pour une utilisation en tant que *compteurs* [41].

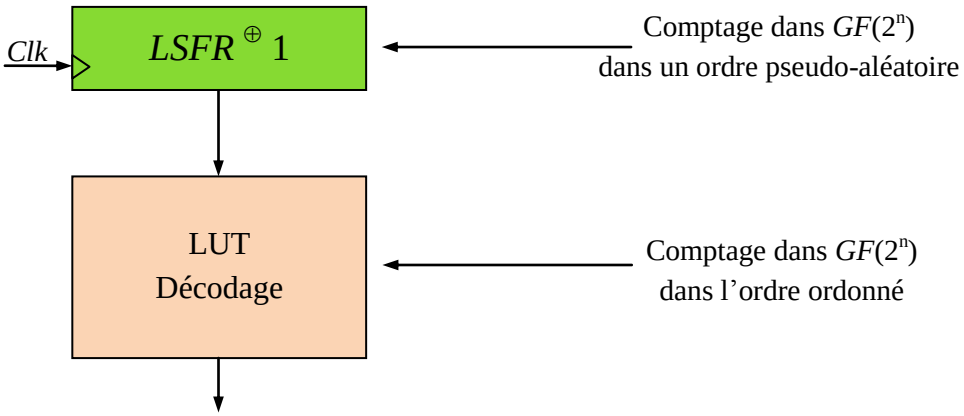


FIG. 3.2 – *Principe d'un compteur basé sur un LFSR*

L'organisation fonctionnelle de l'un de ces compteurs est présentée sur la figure 3.2.

Il est composé de deux parties principales :

- un registre LFSR contrôlé par un polynôme primitif, de degré *n* ;
- une ROM de décodage.

Le rôle de cette ROM est de permettre la conversion entre l'ordre de parcours des éléments de $\mathbb{GF}(2^n)$ et l'ordre «traditionnel» croissant utilisé dans les compteurs classiques.

La seule contrainte pour que ce compteur soit fonctionnel est de choisir un état initial R_0 des cellules du registre différent de l'élément nul. Dans le cas contraire, le contenu du registre n'évolue jamais.

Une architecture respectant ces critères se comporte comme un compteur modulo 2^{n-1}. Ce compteur, si le LFSR utilisé est de type *interne*, a l'avantage de ne pas présenter de contraintes liées aux délais de propagation des retenues affectant les compteurs classiques.

3.2.6. Décompteur apparié

Selon le même principe que les compteurs présentés dans la section précédente, j'ai mis au point à l'aide de LFSR des décompteurs ayant la particularité de passer *exactement* par les mêmes états logiques qu'un compteur donné mais avec un sens de parcours inversé.

En l'occurrence, si un compteur $C^{\oplus}$ passe par les états successifs $R_0, R_1, R_2, \ldots R_{2^{n-1}}$ après avoir été initialisé avec la valeur R_0, le décompteur $C^{\ominus}$ apparié comptera lui de la façon suivante :

$$R_{2^{n-1}}, R_{2^{n-2}}, \ldots, R_0, \tag{3.9}$$

si ses cellules ont été initialisées avec la valeur $R_{2^{n-1}}$.

Pour un LFSR défini par la matrice T de l'équation 3.3, la matrice

$$T^{\ominus} = \begin{bmatrix} 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 1 & 0 & \dots & 0 & 0 \\ c_1 & c_2 & \dots & c_{n-1} & 1 \end{bmatrix} \tag{3.10}$$

est caractéristique du décompteur associé.

En effet, le produit de T par $T^{\ominus}$ conduit à la matrice unité I. Par conséquent, $T^{\ominus}$ est la matrice caractéristique du LFSR décompteur apparié à un LFSR compteur défini par la matrice de l'équation 3.3.

Ce type de LFSR peut être utilisé en complément des LFSR compteurs présentés plus haut.

3.3. Application à la démodulation entièrement numérique de la phase

On se propose d'implanter les bases d'un système capable de traiter les signaux de type M-PSK[1] à l'aide de registres.

Les LFSR n'acceptent que des signaux de contrôle binaires. Or, les signaux que nous nous proposons de démoduler sont échantillonnés et quantifiés : la première étape avant tout traitement est donc la conversion du signal modulé en un signal image présentant la même phase et la même fréquence que le signal original, mais dont l'amplitude ne peut prendre d'autres valeurs que 0 ou 1. En d'autres termes, il s'agit de convertir le signal modulé en un signal purement binaire.

[1] *M-ary Phase Shift Keying - Codage M-aire par Déplacement de Phase*

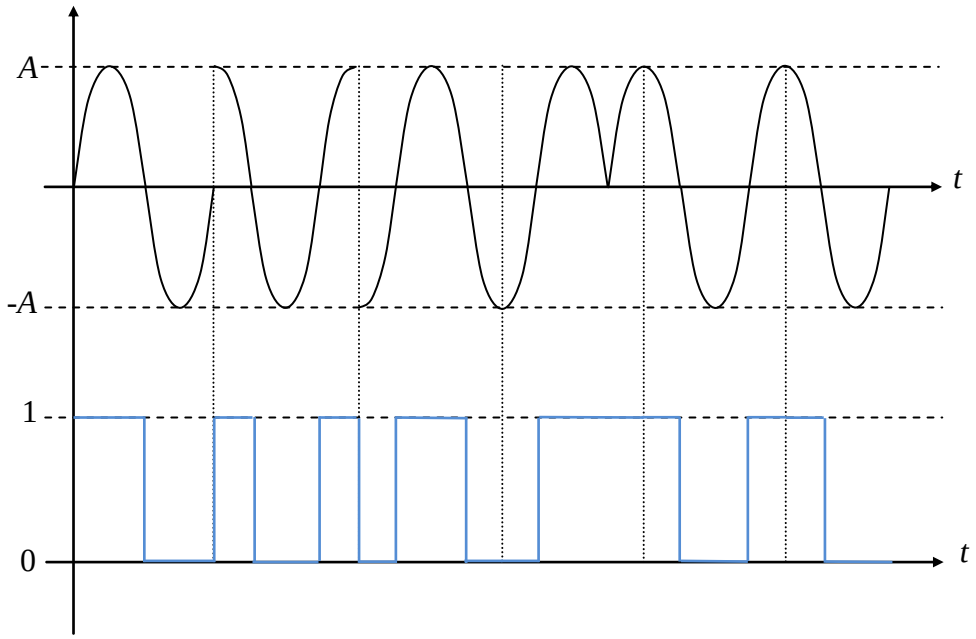


FIG. 3.3 – *Mise en forme d'un signal M-PSK.*

Cette opération est effectuée à l'aide d'un système de détection de seuil intégré au convertisseur analogique numérique présent en amont du circuit numérique sur lequel notre architecture est implantée.

Un signal modulé M-PSK répond à l'équation :

$$S(t) = A\sin(2\pi f \cdot t + \emptyset_q) \tag{3.11}$$

avec $q \cdot \Delta t \leq t < (q+1) \cdot \Delta t$. Comme précédemment, Δt est la durée d'un symbole M-PSK et $\emptyset_q$ la phase associée à ce symbole.

Le signal d'entrée est tout d'abord converti en une forme purement binaire, qui autorise un traitement simplifié sur un circuit numérique à l'aide d'opérations combinatoires simples. La nouvelle équation définissant le signal $S(t)$ après échantillonnage, quantification et cette dernière transformation est :

$$m(n) = \begin{cases} 1, & \text{si } \frac{2k\pi - \emptyset_q}{2\pi f} \leq t < \frac{(2k+1)\pi - \emptyset_q}{2\pi f} \\ 0, & \text{dans les autres cas,} \end{cases} \tag{3.12}$$

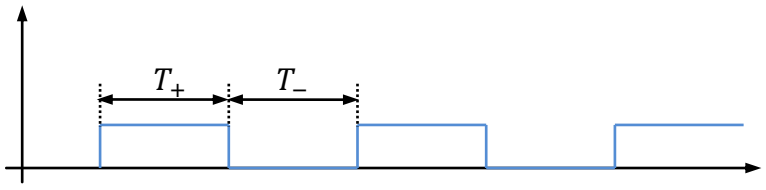
Après transformation $m(n)$ conserve la même fréquence et les mêmes phases que le signal original avec la précision permise par la fréquence d'échantillonnage utilisée.

3.3.1. Architecture du démodulateur

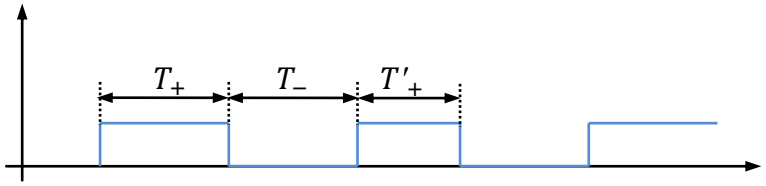
Le démodulateur de phase utilise des registres LFSR fonctionnant en mode compteur ou décompteur pour mesurer la *régularité* du signal $m(n)$ qu'il est chargé de traiter. En d'autres termes un changement de phase sur un signal binaire carré se traduira par une rupture de la régularité des périodes pendant lesquelles le signal est à l'état haut et celles pendant lesquelles il est à l'état bas. Dans le cas d'un signal ne subissant aucun déphasage la durée de ces périodes doit être identique, ou pratiquement identique du fait des artefacts pouvant être introduits par l'opération d'échantillonnage.

Cette différence est présentée sur la figure 3.4. Sur le chronogramme 3.4a, $T_+ = T_-$ le signal conserve la même phase sur toute la durée du graphe. Par contre, sur la figure 3.4b, $T'_+ < T_-$ ce qui dénote un déphasage $\Delta\emptyset$. Celui-ci peut-être évalué de la façon suivante

$$\Delta\emptyset = 2\pi f \cdot (T'_+ - T_-) \tag{3.13}$$



a) Signal non déphasé



b) Signal avec un déphasage

FIG. 3.4 – *Durées des états haut et bas pour un signal non déphasé, et un signal avec un déphasage.*

L'architecture proposée pour notre démodulateur de phase est présentée sur la figure 3.5. Ce démodulateur est composé principalement de quatre registres LFSR, notés *LFSR*⊕ 1, *LFSR*⊕ 2, *LFSR*⊖ 1 et *LFSR*⊖ 2.

Les *LFSR*$^{\oplus}$ 1 et *LFSR*$^{\oplus}$ 2 fonctionnent tous les deux en mode compteur, et partagent le même polynôme générateur. Par conséquent, leur cycle de comptage est identique. La valeur initiale du contenu des cellules de ces deux LFSR est notée R_0, et la valeur après i itérations R_i .

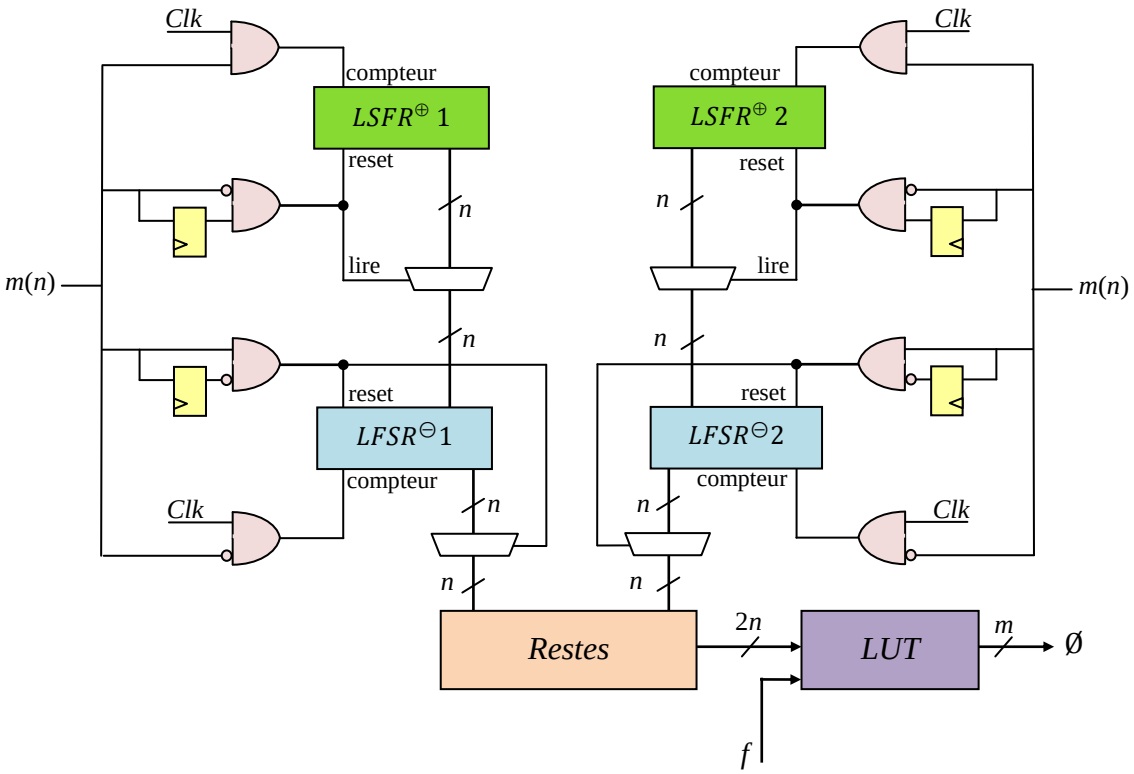


FIG. 3.5 – *Architecture du démodulateur de phase*

Les *LFSR*⊖ 1 et *LFSR*⊖ 2 fonctionnent en mode décompteur. Comme dans le cas des deux LFSR compteurs, ils sont identiques, et partagent le même polynôme générateur. De plus, ils sont *appariés* aux deux LFSR⊕ 1 et LFSR⊕ 2, ce qui signifie qu'à chaque itération, leur contenu sera modifié d'un facteur R_{-1} : si à l'itération i, le contenu de l'un de ces registres est R_k , il sera R_{k-1} après l'itération suivante.

– Le *LFSR*$^{\oplus}$ 1 fonctionne en tant que compteur synchrone. Il n'est incrémenté que dans le cas où $m(n)$ est à l'état logique 1. Dans ce cas, à chaque période T_{clk} de l'horloge interne du circuit numérique sur lequel cette architecture est implantée, *LFSR*$\oplus$ 1 subit une itération. Si $m(n)$ reste à l'état logique 1 pendant k périodes d'horloge, le contenu de ce registre sera :

$$\{LFSR^{\oplus}\ 1\} = R_k. \tag{3.14}$$

Le contenu de ce registre est remis à sa valeur initiale R_0 après détection d'un front descendant de $m(n)$. Ce LFSR permet donc de mesurer la durée pendant laquelle $m(n)$ reste à l'état haut.

– Le *LFSR*$^{\oplus}$ 2 a un comportement exactement similaire à celui de *LFSR*$^{\oplus}$ 1. Simplement, il est décrémenté lorsque $m(n)$ est à un état logique bas, et est remis à son état initial R_0 lors de la détection d'un front montant de $m(n)$. À la différence du premier compteur, celui-ci permet de mesurer la durée d'un état logique bas de $m(n)$, et l'état de ses cellules pour un signal $m(n)$ ayant duré $k \cdot T_{clk}$ est noté $\{LFSR^{\oplus}\ 2\} = R_k$.

– Le *LFSR*$^{\ominus}$ 1 est sensible aux états logiques bas de $m(n)$, tout comme *LFSR*$^{\oplus}$ 2. Mais il s'agit d'un décompteur, qui est *initialisé* avec le contenu du registre *LFSR*$^{\oplus}$ 1 au moment où le signal $m(n)$ passe de l'état haut à l'état bas. Par exemple, si *LFSR*$^{\oplus}$ 1 contenait la valeur R_k juste avant le front descendant dénotant pour $m(n)$ le passage de l'état haut à l'état bas et donc le début du décomptage par *LFSR*$^{\ominus}$ 1, ce dernier sera initialisé à R_k et prendra successivement les valeurs R_k , R_{k-1} , ….

– Le *LFSR*$^{\ominus}$ 2 est un décompteur dont le comportement est proche de celui de *LFSR*$^{\ominus}$ 1. Par contre il réagit aux états logiques hauts de $m(n)$ en lieu et place des états logiques bas et la valeur initiale de ses cellules correspond à celle des cellules de *LFSR*$^{\oplus}$ 2 juste avant un front montant de $m(n)$.

3.3.2. Mesure du déphasage

Un déphasage est détecté si, avec l'une des deux paires de compteurs/décompteurs, une irrégularité dans la durée des périodes à l'état haut ou à l'état bas du signal $m(n)$ est constatée.

Pour le premier couple de compteurs/décompteurs, une période de $m(n)$ commence par un front montant indiquant le passage de ce signal à l'état haut. Cette période se termine au front montant suivant. Si aucun déphasage n'intervient pendant un cycle de comptage sur une période de $m(n)$ dont la durée est $2 * k$ cycles :

- $LFSR^{\oplus}$ 1 compte de R_1 à R_k pendant la demi-période où $m(n)$ est à l'état haut ;
- $LFSR^{\ominus}$ 1 décompte de R_k à R_1 pendant la demi-période où $m(n)$ est à l'état bas.

Il y a déphasage si le contenu de $LFSR^{\ominus}$ 1 est différent de R_1 au moment du front montant de $m(n)$ marquant le début d'une nouvelle période.

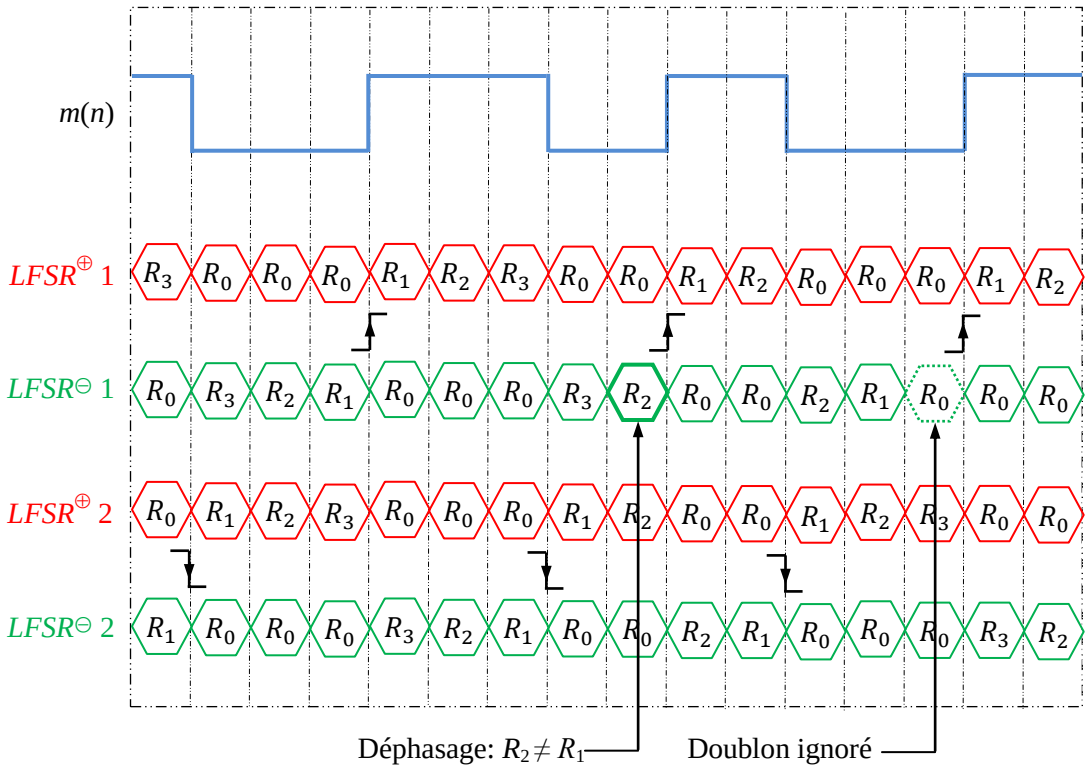


FIG. 3.6 – *Détection d'un déphasage - premier chronogramme*

Le comportement de la seconde paire de compteur est symétrique de celui décrit ci-dessus: pour cette paire, une période de $m(n)$ est délimitée par deux fronts descendants au lieu de deux fronts montants. Là aussi, si le contenu de $LFSR^{\ominus}$ 2 n'est pas égal à R_1 à la fin d'une période, un déphasage a été détecté.

Les figures 3.6 à 3.8 présentent les trois cas de détection d'un déphasage.

Un exemple d'irrégularité ne pouvant être détectée que par la paire de compteurs ($LFSR^{\oplus}1$, $LFSR^{\ominus}1$) est présenté sur la figure 3.6. Au moment du second front montant de $m(n)$ le contenu de $LFSR^{\ominus}$ 1 est R_2, ce qui indique un déphasage. Le même déphasage est détecté par la paire de compteurs 1 au moment du front montant suivant : il sera ignoré par la logique de décodage de l'architecture présentée sur la figure 3.5.

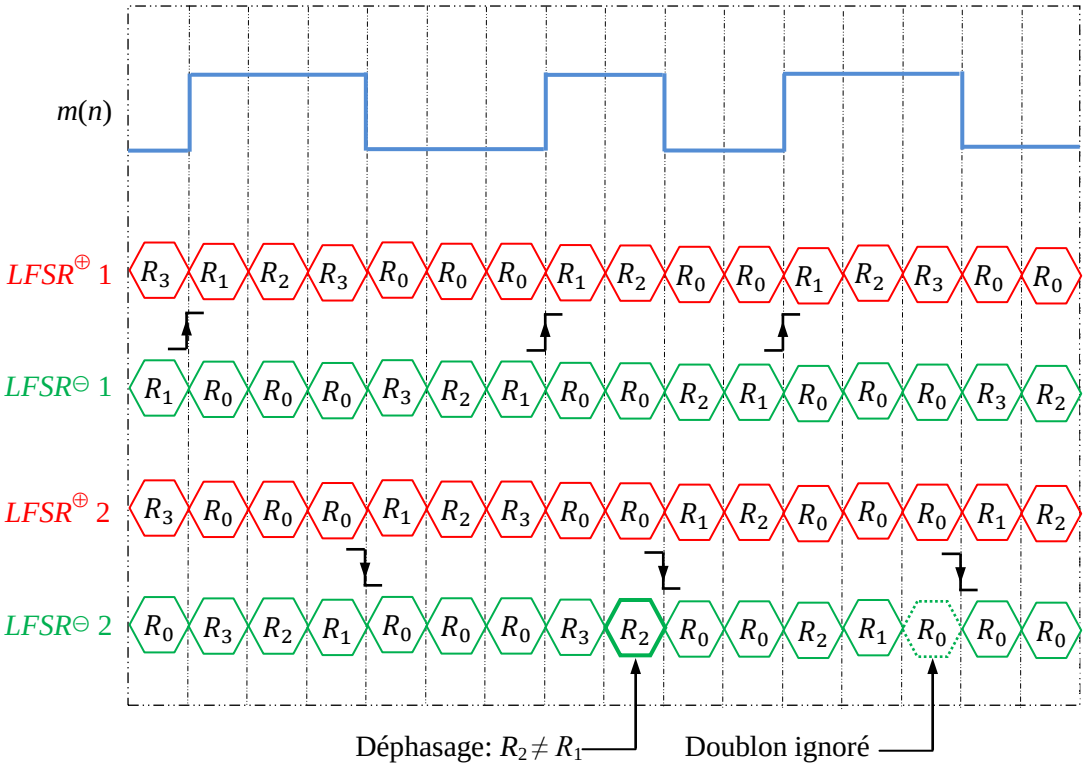


FIG. 3.7 – *Détection d'un déphasage - second chronogramme*

Le déphasage présenté sur la figure 3.7 ne peut être détecté que par la seconde paire de compteurs, puisque la première ne perçoit pas d'irrégularité dans la succession d'états hauts et bas de $m(n)$. À l'instant du second front descendant de $m(n)$ le

contenu des cellules de $LFSR^{\ominus}$ 2 est R_2 au lieu de R_1. Un déphasage est donc détecté à ce moment. Comme dans le cas précédent, la seconde détection du même déphasage est ignorée par la logique de décodage.

Enfin, le déphasage présenté sur la figure 3.8 est détecté tout d'abord par la seconde paire de compteurs/décompteurs, puis par la première. Là encore, la seconde détection n'est pas prise en compte par le décodeur.

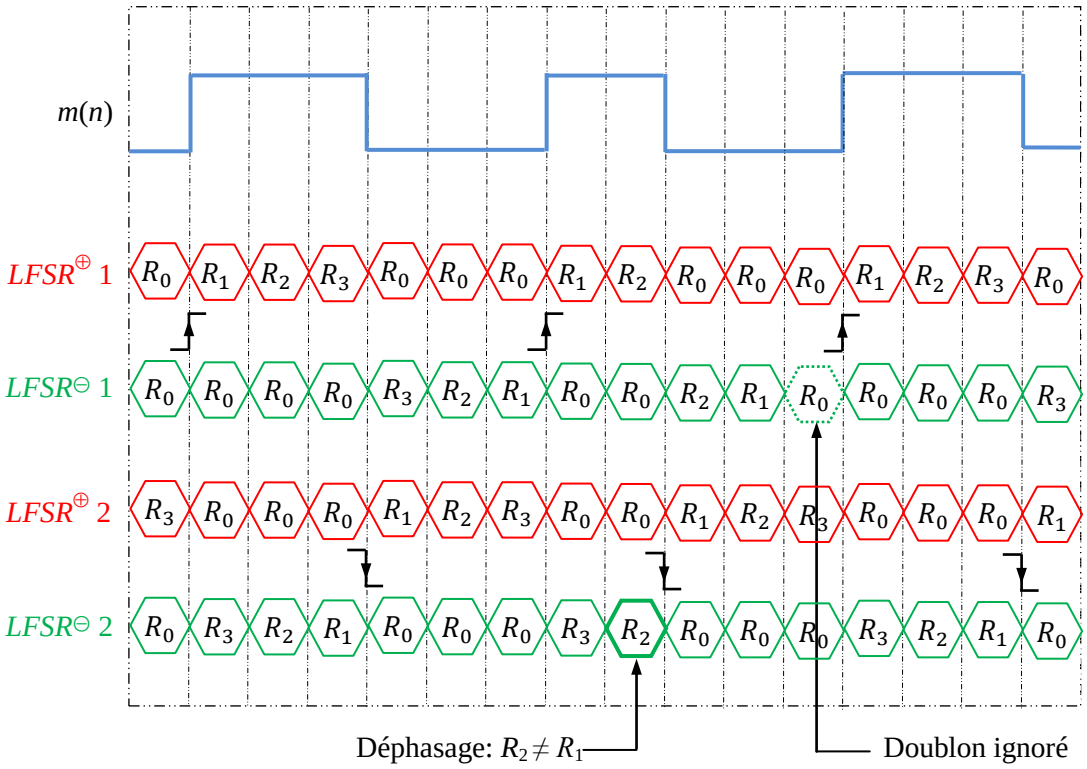


FIG. 3.8 – *Détection d'un déphasage - troisième chronogramme*

Lorsqu'un déphasage est détecté le contenu des cellules de $LFSR^{\ominus}1$ et $LFSR^{\ominus}2$ constituant les «restes» du décomptage permettent de calculer le déphasage de $m(n)$, à l'aide de l'équation 3.13. Ce déphasage dépend encore de la fréquence du signal $m(n)$. La conversion finale est effectuée par une LUT contenant la table de conversion entre les couples de valeurs (R_i, f) et la phase $\emptyset$.

3.4. Conclusion

Dans ce chapitre, nous avons présenté une nouvelle architecture de démodulation numérique de signaux modulés en phase. Elle utilise deux des compteurs efficaces, basés sur des registres LFSR, dont la dimension et le polynôme caractéristique permettent de contrôler la durée du comptage.

Une méthode pour créer un décompteur apparié à un compteur donné a été proposée et mise en application pour réaliser une architecture pour la démodulation de phase, utilisant quatre LFSR compteurs/décompteurs. Cette architecture atteint des performances meilleures que les architectures existantes et occupe un nombre restreint de cellules logiques.

Les LFSR sont des systèmes particulièrement bien adaptés à la structure interne des cellules logiques élémentaires de processeurs FPGA. Leur utilisation dans cette architecture de démodulation a permis d'obtenir de bonnes performances sur ce type de circuit, ce qui démontre l'intérêt d'une méthodologie de conception prenant en compte très en amont la technologie cible.

Chapitre 4

Simulation & Synthese

4.1. Introduction

Ce quatrième et dernier chapitre représente la partie de la validation de notre projet qui consiste à la simulation et l'implémentation de circuits de modulateur et démodulateur, ces derniers sont réalisés à l'aide de circuit FPGA.

Les sections suivantes décrivent la mise en œuvre de modulateur et démodulateur sur le circuit logique programmable FPGA. Dans notre travail, on a utilisé une carte FPGA Virtex II XC2V1000 disposant d'un signal d'horloge de fréquence 100MHz.



FIG 4.1– *La carte FPGA Virtex II XC2V1000*

Pour la programmation du code VHDL, le logiciel Xilinx ISE 8.2i a été utilisé. Ce logiciel permet de compiler, de simuler, de synthétisé et de générer le fichier VHDL (bistream). Le bistream a été programmé sur le FPGA via le câble JTAG. Afin d'acquérir et visualiser les différents signaux à la sortie da notre carte FPGA.

La partie validation est organisée en deux sections. Nous présenterons dans la première les résultats de simulation et dans la deuxième les résultats de synthèse de modulateur et démodulateur.

4.2. Résultats de simulation de modulateur/démodulateur

Plusieurs simulations ont été réalisées à différentes étapes du flot de conception sur FPGA, à savoir des simulations RTL[1] au niveau porte, des simulations post-synthèse et finalement post-placement-routage. Ceci nous a permis de vérifier le bon fonctionnement du modulateur et démodulateur à différentes étapes du prototypage et ainsi de valider l'architecture proposée.

Nous commencerons la phase de vérification du code VHDL (voir l'Annexe B.1) de modulateur par simulation en isolant chaque composant et en le stimulant par un testbench.

- ***Pré-codeur***

L'étage pré-codeur constitue le premier bloc que traverse l'information au niveau de Modulateur. Pour tester son fonctionnement, nous avons créé un testbench qui présente au niveau de l'entrée `entree_pr_cod`, la séquence binaire suivante `00110010` en choisissons le types de modulation `type_modu` de bits de poids fort `10`, c'est-à-dire on applique le principe de la modulation QAM16 :

1. Nous commençons par lire l'entrée par groupe de huit bits, nous avons donc par exemple le mot suivant : `00110010`.
2. Nous obtiendrons à sortie:

[1] *Register Transfer Level*

`sortir_pr_cod=00000010.`

Le résultat de simulation de l'étage « Pré-codeur» est présenté dans la figure 4.2

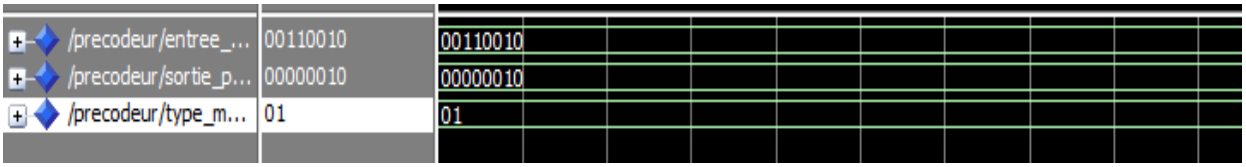


FIG. 4.2– *Simulation de l'étage «Pré-codeur».*

- ***Codeur***

Nous récupérons un autre sortie `sortir_pr_cod=00110010` par exemple et nous créons un autre testbench pour simuler le fonctionnement du bloc «Codeur» et nous appliquons eu entré `entree_cod` la sortie qu'on vient de récupérer du premier testbench. Le résultat de cette simulation comme le montre la figure 4.3 correspond bien les valeurs de `rho=11101111` et `theta=10000000` qui sont déjà chargés dans les ROMs, le `post_rotation=00` indique que nous travaillons dans le premier quadrant.

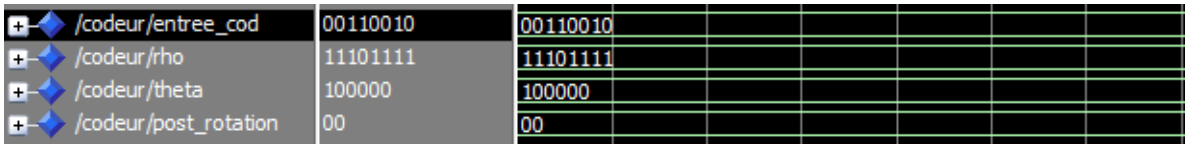


FIG. 4.3– *Simulation de l'étage «Codeur».*

- ***Générateur de sinusoïdes (Processeur CORDIC)***

Comme dans les étages précédents la simulation de ce processeur CORDIC qui a une structure itérative sous la forme d'un pipeline est effectuée sous ModelSim. La figure 4.4 est la simulation testbench du processeur CORDIC, l'horloge de simulation fonctionne à 100 MHz. Il montre comment la sortie (valeurs sinus et cosinus) converge

vers une valeur particulière avec 8 itérations. Il montre également les différents signaux intermédiaires comme `X1, Y1, Z1`, …etc.

Les entrées de processeur CORDIC sont le signal d'horloge `clk`, le signal remise à zéro `rst` (reset), le signal `ce` pour contrôler l'ensemble du module à travailler ou non, et `theta, rho_2` sont l'angle et l'amplitude respectivement du signal en entrée la taille de ces signaux est de 8 bits représentés en complément à deux, à la sortie de processeur CORDIC on obtient les signaux `sin` et `cos` de phase `theta` en radian de taille 8 bits.

On peut émettre plusieurs remarques suite à la simulation de ce bloc.

- A chaque étape, les décalages sont toujours d'amplitude fixe, car ils ne dépendent que de l'ordre (appelé `cell_id` dans la description VHDL) de la tranche du pipeline.
- Les valeurs des $arctan\ 2^{-i}$ sont pré calculées et stockées dans une structure de type `array`, et fournies par la suite à chaque cellule, la taille de chaque valeur est 9 bits représenté en complément à deux.
- L'angle en entrée appelé `theta` correspond à la résultante de la somme $\omega t + \theta$, où θ est l'angle de rotation souhaitée.
- `rho_2` est l'amplitude du signal de sortie, pondérée pour compenser le facteur d'échelle.

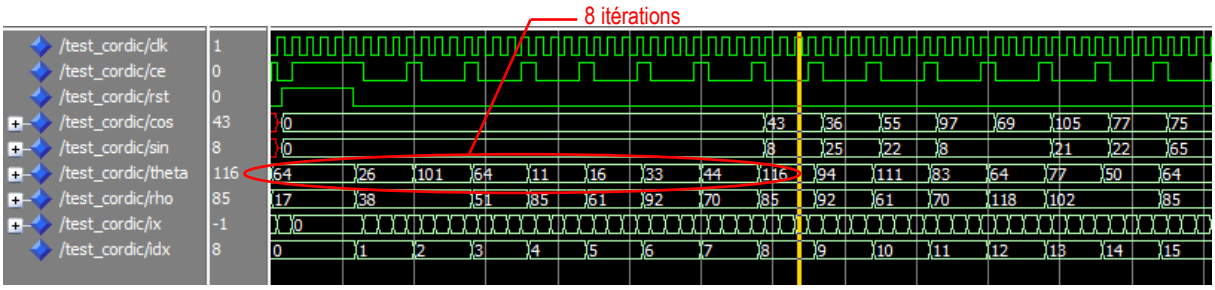


FIG. 4.4– *Simulation de processeur CORDIC*

Les premières valeurs de tests sont :

`x0=17, y0=0` et `z0=64`, (les valeurs en décimal)

Nous devons retrouver en sortie les valeurs `cos=43` et `sin=8`, en ajoutant ici une remarque telle que les valeurs précédentes sont des résultats multiplié avec l'amplitude `rho_2`.

- ***Bloc d'échantillonnage des signaux***

Puisque nous cherchons à produire une sinusoïde, ce bloc permet d'obtenir, pour chaque symbole à émettre présenté à l'entrée du ce bloc, plusieurs échantillons à la sortie, pour tester son fonctionnement, nous avons créé un testbench qui présente au niveau de l'entrée `cos`, `sin` et le quadrant (post rotation) `ps_rot`, on trouve à la sortie le signal de 4 échantillons `IQ`. Le résultat de simulation de l'étage «Echantillonnage» présenté dans la figure 4.5

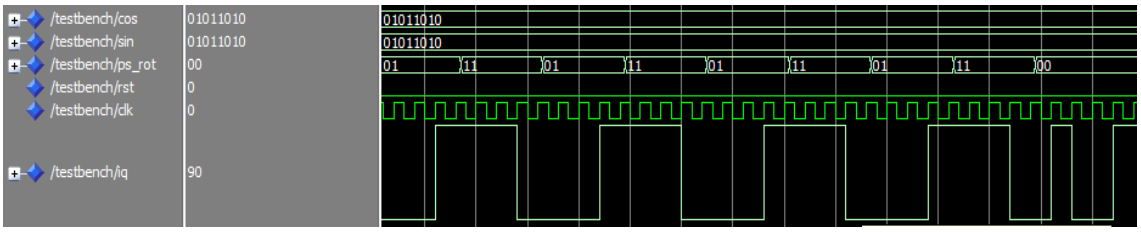


FIG. 4.5 – *Simulation de l'étage «Bloc d'échantillonnage».*

Par exemple si l'horloge de simulation $clk = 20ns$ donc le période d'un seul échantillon est $T_e = 20ns$ et le fréquence d'échantillonnage $f_{ech} = 1 / T_e$, on peut dire que la fréquence de porteuse est $f_p = 1/4\ f_{ech}$.

- ***Démodulateur***

Avant d'exposer le résultat de simulation du code VHDL responsable de démodulateur de type 4PSK (voir l'Annexe B.2), et de l'insertion des symboles, nous rappelons le principe de fonctionnement de ce bloc :

- L'entrée `data` de ce bloc est un symbole de données constitué par plusieurs bits successifs.
- On choisit une période qui contient 6 cycles d'horloge.
- La LUT contenant la table de conversion entre les couples de valeurs contenu des cellules du dernier LFSR (Reste) et le déphasage en fonction de la fréquence de signal d'entrée.
- La sortie de ce bloc est donc un symbole de taille 2 bits (représente les quadrants de la constellation).

Pour simuler le fonctionnement de ce bloc, nous avons créé un testbench qui applique au bloc `lfsrdemo` un signal binaire. La figure 4.6 présente le résultat de simulation.

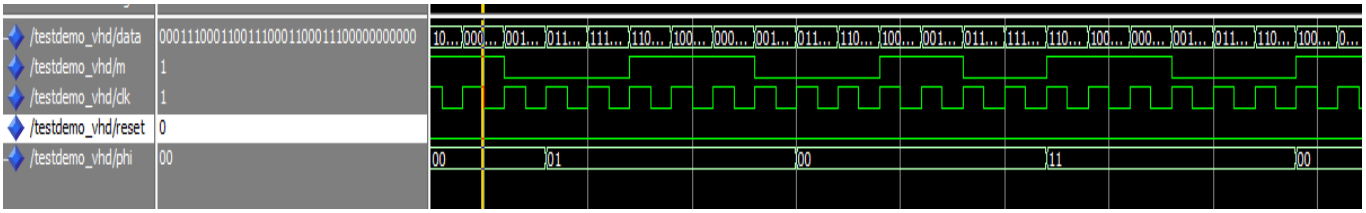


FIG. 4.6 – *Simulation de démodulateur.*

4.3. Résultats de synthèse de modulateur/démodulateur

Nous avons effectué à synthèse en prenant en considération ce qui suit :

- D'abord notre système demande des ressources disponibles dans le circuit Virtex II.
- La taille de mémoire ROMs pour les tableaux pré-calculés disponible sous le circuit Virtex II.

Nous allons donc réaliser la synthèse de modulateur et de démodulateur séparément et nous exploiterons les rapports de consommation des ressources et le 'Timing Report' pour dégager les délais ou retard générés par notre code. Les tableaux générés par Xilinx ISE pour résumer la synthèse des composants ont été recopiés dans l'annexe C.

- ***Synthèse de modulateur***

Avant l'opération de synthèse de modulateur nous allons câbler d'abord leurs composants.

D'après l'Annexe C.1, le bloc modulateur occupe 199 Slices parmi les 5120 Slices offertes par le circuit Virtex II soit une occupation de 3%.

Le rapport de synthèse du bloc modulateur nous a permis de dégager la valeur de la période minimale, elle est de 6.176 ns, ce qui fait que ce bloc peut fonctionner à la fréquence maximale : f_{max}=161.970 MHz.

La figure 4.7 illustre le schéma RTL du bloc modulateur vu de l'extérieur et le schéma Netlist de ce bloc disponible dans l'Annexe D.1.



FIG. 4.7 – *Schéma RTL externe du bloc 'modulateur'*

- ***Synthèse de démodulateur***

Après sa synthèse (Annexe C.2), le bloc démodulateur consomme 29 Slices parmi les 5120 disponibles c'est-à-dire 1% des ressources du circuit. Sa période minimale de fonctionnement étant 3.105 ns, donc ce bloc peut fonctionner à une fréquence maximale : f_{max} = 322.061 MHz.

Le schéma RTL externe est décrit dans la figure 4.9 et le schéma Netlist de ce bloc disponible dans l'Annexe D.2.



FIG. 4.9 – *Schéma RTL externe du bloc 'démodulateur'*

Nous récapitulons dans le tableau 4.1 les résultats de synthèse des différents blocs du modulateur/démodulateur.

Désignation	***Occupation en termes de ressources***	***Fréquence maximale possible***
Modulateur	199 slices et 2 ROMs	161.970MHz
Démodulateur	29 slices et 1 ROM	322.061MHz

TAB 4.1 – *Récapitulation des résultats de synthèse.*

4.4. Conclusion

La phase de validation de l'implémentation est une phase importante et demande beaucoup plus de temps que les autres phases du travail. Elle permet de détecter et de corriger les erreurs grâce à la simulation.

Dans ce dernier chapitre nous avons simulé les blocs constituants le modulateur et le démodulateur implémenté, cette simulation a permis de confirmer la validité du code implémenté. En deuxième lieu, nous avons présenté le résultat de synthèse des différents étages de notre Modem.

D'après la comparaison entre les résultats d'implantation de notre architecture de modulateur QAM et l'architecture à base de ROM on peut remarquer que en terme de fréquence d'échantillonnage de la sinusoïde, les architectures à base de ROM est plus performante que l'architecture CORDIC pipeline, mais en terme de mémoire on constate que l'occupation en surface de l'architecture à base de ROM augmente très rapidement, la conséquence directe de cette augmentation est la chute très importante de la vitesse de fonctionnement du circuit. Les architectures à base de processeurs

CORDIC sont les seules permettant d'atteindre des fréquences d'échantillonnage acceptables sur FPGA.

Les résultats de l'implantation de l'architecture de démodulation de phase PSK sur un FPGA XC2V1000 sont présentés dans cette section. Ils sont comparés avec les performances d'autres architectures numériques de démodulation présentes dans la littérature, qui utilisent un oscillateur contrôlé numériquement pour supprimer la fréquence porteuse du signal modulé qui utilise un système de sur échantillonnage.

Notre architecture à base de LFSR permet d'obtenir une fréquence supérieure aux deux autres architectures testées au delà d'une précision de 8 bits; notre architecture à reste la plus efficace en termes de fréquence en fonction de la surface que les autres architectures numériques de démodulation.

Conclusion générale

Dans le présent memoire, Nous avons étudié et examiné plusieurs méthodes permettant la réalisation des blocs fonctionnels principaux d'un modulateur entièrement numérique. La méthodologie de conception que nous avons décrite dans ce mémoire s'efforce de s'affranchir des difficultés posées par le passage du domaine analogique au domaine numérique. Elle propose des architectures traduisant directement les fonctionnalités nécessaires à la modulation et la démodulation de signaux en algorithmes adaptés aux circuits numériques, et aux FPGA[1] en particulier.

Notre modulateur QAM[2], utilisant en son cœur un algorithme de type CORDIC[3] modifié et parallélisé, a été décrit en VHDL et implanté sur un circuit FPGA en utilisant cette nouvelle méthodologie de conception. Ses performances sont jusqu'à huit fois supérieures à celles des architectures analogiques directement transposées en numérique. Notre démodulateur de phase qui utilise plusieurs bancs de registres de type LFSR[4], atteint des performances du même ordre de grandeur.

Ces deux systèmes ont été conçus de façon à tirer au mieux parti de la structure interne spécifique des circuits FPGA de type XC2V1000 sur lesquels ils sont implantés. En conséquence, le circuit peut fonctionner à une fréquence d'horloge qui s'approche de la fréquence maximale donnée par le constructeur.

En conclusion, nous avons démontré dans ce memoire la faisabilité de l'implantation numérique d'algorithmes de traitement du signal utilisés pour les transmissions à haut débit, en nous focalisant plus particulièrement sur les étapes de modulation et de démodulation. Nous avons défini une méthodologie de conception, qui permet de transcrire les fonctions composant ces étapes en algorithmes optimisés pour une implantation sur un circuit numérique de type FPGA.

---

[1] *Field-Programmable Gate Array*
[2] *Quadrature Amplitude Modulation*
[3] *COordinate Rotation DIgital Computer*
[4] *Linear Feedback Shift Register*

Annexes

Annexe A

Les matrices des amplitudes et des phases de QAM 256

A.1. MATRICE DES AMPLITUDES

Pour calculer l'ensemble des amplitudes possibles à chaque point de la constellation QAM 256, on peut utiliser un petit script sous logiciel SCILAB, qui génère automatiquement une matrice contentant les valeurs nécessaires

```
n=8 // QAM 256
// Matrice des amplitudes
    a=zeros(n,n);
    for k=1:n
    for l=1:n
    a(k,l)=sqrt((2*k-1)^2+(2*l-1)^2);
    end
end
// Matrice des amplitudes approchées
a_max=max(a);
a_res=n; //resolution en nombre de bits
a_step=a_max/(2^a_res-1);
round(a/a_step);
dec2bin(round(a/a_step))
ans=
!00010001 00100110 00111101 01010101 01101101 10000101 10011101 10110101!
!00100110 00110011 01000110 01011100 01110010 10001001 10100000 10111000!
!00111101 01000110 01010101 01100111 01111100 10010001 10100111 10111110!
!01010101 01011100 01100111 01110111 10001001 10011101 10110001 11000111!
!01101101 01110010 01111100 10001001 10011001 10101011 10111110 11010010!
!10000101 10001001 10010001 10011101 10101011 10111011 11001101 11100000!
!10011101 10100000 10100111 10110001 10111110 11001101 11011101 11101111!
!10110101 10111000 10111110 11000111 11010010 11100000 11101111 11111111!
```

A.2. MATRICE DES PHASES

De même procédure on peut calculer automatiquement le matrice des phases, nous allons donc choisi de rédiger un petit script SCILAB permettant de génère cette matrice contentant les valeurs des phases correspondants les point de constellation de QAM256

```
n=8 // qam256
// Matrice des phases
b=zeros(n,n);
     for k=1:n
     for l=1:n
     b(k,l)=atan((2*l-1)/(2*k-1));
     end
end
// Matrice approchée des phases
b_max=%pi/2; // valeur de l'angle maximal encodé.
// intéressant à fixer à pi/2 ou pi/4, pour
// faciliter la prise
// en compte des rotations à effectuer plus tard.
b_res=7; // resolution+1 en nombre de bits
// (+1 pour prendre en compte le %pi/2)
b_step=b_max/(2^b_res-1);
round(b/b_step);
dec2bin(round(b/b_step))
ans=
!1000000  1100101  1101111  1110100  1110110  1111000  1111001  1111010!
!0011010  1000000  1010011  1011110  1100101  1101001  1101101  1101111!
!0010000  0101100  1000000  1001101  1010110  1011101  1100001  1100101!
!0001011  0100001  0110010  1000000  1001010  1010001  1010111  1011100!
!0001001  0011010  0101001  0110101  1000000  1001000  1001110  1010011!
!0000111  0010110  0100010  0101110  0110111  1000000  1000110  1001100!
!0000110  0010010  0011110  0101000  0110001  0111001  1000000  1000101!
!0000101  0010000  0011010  0100011  0101100  0110011  0111010  1000000!
```

Annexe B

B.1. Description des blocs de modulateur

B.1.1. Paramétrage

```
----------------------
-- parametres généraux
----------------------
library ieee;
use ieee.std_logic_1164.all;
package parametres is
-- taille du drapeau 'type de modulation'
constant type_modu_n: positive :=2;
constant qpsk: std_logic_vector :="00";
constant qam16: std_logic_vector := "01";
constant qam64: std_logic_vector := "10";
constant qam256: std_logic_vector := "11";
-- taille de la trame QAM en entrée
constant qam_n: positive :=8;
-- résolution de l'amplitude
constant rho_n: positive :=8;
-- résolution de l'angle
constant theta_n: positive :=9;
-- tables précalculées pour l'amplitude et l'angle
type liste_rho is
array (0 to (2**(qam_n-2)-1)) of std_logic_vector
(rho_n-1 downto 0);
type liste_theta is
-- les 2 MSBs représentent le quadrant et ne sont pas stockés
array (0 to (2**(qam_n-2)-1)) of std_logic_vector
(theta_n-3 downto 0);
constant rho_tab: liste_rho:=
("00010001","00100110","00100110","00110011","01010101","00111101",
"01011100","01000110","01010101","01011100","00111101","01000110",
"01110110","01100110","01100110","01010101","10110101","10011101",
"10111000","10100000","01101101","10000101","01110010","10001001",
"11000111","10110001","10111110","10100111","10001001","10011101",
"01111100","10010001","10110101","10111000","10011101","10100000",
```

```
"11000111","10111110","10110001","10100111","01101101","01110010",
"10000101","10001001","10001001","01111100","10011101","10010001",
"11111111","11101111","11101111","11011101","11010010","11100000",
"10111110","11001101","11010010","10111110","11100000","11001101",
"10011001","10101011","10101011","10111011",
others => "00000000");
constant theta_tab: liste_theta:=
("1000000","0011010","1100101","1000000",
"0001011","0010000","0100001","0101100",
"1110100","1011110","1101111","1010011",
"1000000","1001101","0110010","1000000",
"0000101","0000110","0010000","0010010",
"0001001","0000111","0011010","0010110",
"0100011","0101000","0011010","0011110",
"0110101","0101110","0101001","0100010",
"1111010","1101111","1111001","1101101",
"1011100","1100101","1010110","1100001",
"1110110","1100101","1111000","1101001",
"1001010","1010110","1010001","1011101",
"1000000","1000101","0111010","1000000",
"1010011","1001100","1001110","1000110",
"0101100","0110001","0110010","0111001",
"1000000","0110111","1001000","1000000",
others => "0000000");
end parametres;
```

B.1.2. PRECODEUR

```
-------------
-- pré codeur
-------------
library ieee;
use ieee.std_logic_1164.all;
use work.parametres.all;
entity precodeur is
port (
            type_modu : in std_logic_vector(type_modu_n-1 downto 0);
            entree_pr_cod : in std_logic_vector (qam_n-1 downto 0);
            sortie_pr_cod : out std_logic_vector (qam_n-1 downto 0)
```

```
      );
end precodeur;
architecture arch_precodeur of precodeur is
begin
with type_modu select
 sortie_pr_cod <= entree_pr_cod(0) & entree_pr_cod(1) & "000000" when
qpsk,
 entree_pr_cod(3 downto 2) & "0000" & entree_pr_cod(1 downto 0) when
qam16,
 entree_pr_cod(5 downto 4) & "00" & entree_pr_cod(3 downto 0) when qam64,
 entree_pr_cod when qam256,(others => '0') when others;
end arch_precodeur;
```

B.1.3. CODEUR

```
---------
-- codeur
---------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use work.parametres.all;
entity codeur is
      port (entree_cod : in std_logic_vector (qam_n-1 downto 0);
      theta : out std_logic_vector (theta_n-3 downto 0);
      post_rotation : out std_logic_vector (1 downto 0);
      rho : out std_logic_vector (rho_n-1 downto 0));
end codeur;
architecture arch_codeur of codeur is
begin
rho <= rho_tab(conv_integer(entree_cod((qam_n-3) downto 0)));
theta(theta_n-3 downto 0) <= theta_tab(conv_integer(entree_cod((qam_n-4)
downto 0)));
-- rotation de quadrant
post_rotation(1 downto 0) <= entree_cod((qam_n-1) downto (qam_n-2));
end arch_codeur;
```

B.1.4. PROCESSEUR CORDIC A ACCUMULATEUR

```
--------------------------------------------------
-- Descreption VHDL
-- "Cordic à accumulateur"
-- Mr Boucherit Toufik, Université de Médéa, Algérie
--------------------------------------------------
-- force clk 0 0, 1 40 -repeat 80
-- force rst 1 0, 0 60
-- run 300 ns
----------------------
-- Constantes CORDIC
----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
package constantes_cordic is
constant out_n: integer :=8;
constant m: integer :=1;
constant pipeline_n : natural := 8;
signal count_s : integer range 0 to 7 ;
end constantes_cordic;
---------------------------------------
-- Description d'une cellule du Pipeline
---------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use work.constantes_cordic.all;
entity cordic_cell is
generic (resolution : natural := 8;
cell_id : natural := 1
);
port
 (clk,ce ,rst : in std_logic;
-- les 'signed' sont représentés en
-- complément à 2
X1 : in signed(resolution-1 downto 0);
Y1 : in signed(resolution-1 downto 0);
Z1 : in signed(resolution-1 downto 0);
X2 : out signed(resolution-1 downto 0);
Y2 : out signed(resolution-1 downto 0);
Z2 : out signed(resolution-1 downto 0)
);
end entity cordic_cell;
architecture arch_cordic_cell of cordic_cell is
```

```
---------------------
-- functions
---------------------
function arctan(n :natural) return signed is
  variable result: signed(resolution-1 downto 0);
  begin
    case n is
      when 0 => result := "1000000";
      when 1 => result := "0100101";
      when 2 => result := "0010100";
      when 3 => result := "0001010";
      when 4 => result := "0000101";
      when 5 => result := "0000011";
      when 6 => result := "0000001";
      when 7 => result := "0000001";
   when others => result := "0000000";
    end case;
        return result;
  end arctan;
-- fonction décalage à droite arithmétique
-- d'amplitude N ( sur entier en complément à 2)
function rshift(val : in signed) return signed is
variable tampon : signed(val'range);
begin
for i in val'high downto (val'high-cell_id+1) loop
-- on place à 0 (nbr positif) ou 1 (nbr négatif)les n MSBs
tampon(i):= val(val'high);
end loop;
for i in (val'high-cell_id) downto 0 loop
-- on décale vers la droite le reste des bits
tampon(i):= val(cell_id+i);
end loop;
return tampon;
end function rshift;
-- fonction addition / soustraction
function addsub(v1 : in signed;
v2 : in signed;
sign : in std_logic)
return signed is
begin
if sign = '1' then
return v1+v2;
else
return v1-v2;
end if;
end function addsub;
signal X2_t,Y2_t,Z2_t,X1_s,Y1_s,atan : signed (resolution-1 downto 0);
signal Z_sign : std_logic;
begin
Z_sign <= Z1(resolution-1);
```

```
atan <= conv_signed(arctan(cell_id),8);
--L'angle ne peut pas être négatif,
--'atan' ne renvoie jamais une valeur négative, ainsi conv_signed peut
--être employé
X1_s <= rshift(X1);
Y1_s <= rshift(Y1);
X2_t <= addsub(X1,Y1_s,Z_sign);
Y2_t <= addsub(Y1,X1_s,not Z_sign);
Z2_t <= addsub(Z1,atan,Z_sign);
cordic_process : process(rst,clk)
begin
if rst='1' then
X2 <= (others=>'0');
Y2 <= (others=>'0');
Z2 <= (others=>'0');
else if rising_edge(clk)and ce='1' then
X2 <= X2_t;
Y2 <= Y2_t;
Z2 <= Z2_t;
end if;
end if;
end process cordic_process;
end arch_cordic_cell;
-- Description du Pipeline
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.constantes_cordic.all;

entity cordic is
    generic (pipeline_n : natural := 8);
    port (clk,ce, rst : in std_logic;
    theta : in signed(out_n-1 downto 0);
    rho_2 : in signed(out_n-1 downto 0);
    cos : out signed(out_n-1 downto 0);
    sin : out signed(out_n-1 downto 0)
    );
end entity cordic;
architecture pipeline_simple of cordic is
    type pipeline_array is array (pipeline_n downto 0) of
    signed (out_n-1 downto 0);
-- signaux de connexion entre les différentes cellules du pipeline
    signal X,Y,Z : pipeline_array;
-- cellule de pipeline
component cordic_cell is
    generic (resolution : natural := out_n;
              cell_id : natural := 1
               );
     port (clk,ce, rst : in std_logic;
     X1 : in signed(resolution-1 downto 0);
     Y1 : in signed(resolution-1 downto 0);
```

```
     Z1 : in signed(resolution-1 downto 0);
     X2 : out signed(resolution-1 downto 0);
     Y2 : out signed(resolution-1 downto 0);
     Z2 : out signed(resolution-1 downto 0)
     );
end component cordic_cell;
begin
   -- correspondance des signaux
      X(0)<= rho_2;
      Y(0)<= (others => '0');
      Z(0)<= theta;
      cos <= X(pipeline_n);
      sin <= Y(pipeline_n);
   fabric_pipel:
      for n in 1 to pipeline_n generate
      pipe_cell : cordic_cell
      generic map (resolution => out_n,
         cell_id => n)
         port map (clk => clk,
         rst => rst,ce => ce,
         X1 => X(n-1),
         Y1 => Y(n-1),
         Z1 => Z(n-1),
         X2 => X(n),
         Y2 => Y(n),
         Z2 => Z(n));
   end generate fabric_pipel;
end pipeline_simple;
```

B.1.5. SYNCHRONISATION

```
----------------------
-- package definitions
----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
package definitions is
constant resolution: integer :=8;
constant in_size: integer :=8;
type ech_array is array (0 to 3) of std_logic_vector(in_size-1 downto 0);
end definitions;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use work.definitions.all;
entity echant is
generic (in_size : natural := resolution);
port (clk, rst : in std_logic;
```

```
cos, sin : in std_logic_vector(in_size-1 downto 0);
e : out ech_array);
end echant;
architecture synchro of echant is
signal etage0, etage1 : ech_array;
begin
--etage d'entrée
etage0(3) <= cos;
etage0(2) <= sin;
etage0(1) <= cos;
etage0(0) <= sin;
--étage de sortie
e <= etage1;
synchrone : process(rst,clk)
begin
if rst='1' then
etage1(3) <= (others=>'0');
etage1(2) <= (others=>'0');
etage1(1) <= (others=>'0');
etage1(0) <= (others=>'0');
elsif rising_edge(clk) then
--premier étage
etage1(3) <= etage0(3);
etage1(2) <= -etage0(2);
etage1(1) <= -etage0(1);
etage1(0) <= etage0(0);
end if;
end process;
end synchro;
```

B.2. Description de démodulateur

```
--------------------
-- Compteur
--------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity cntup is
 Port ( clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        ce : in  STD_LOGIC;
        Q : out  STD_LOGIC_VECTOR (3 downto 0));
end cntup;
architecture Behavioral of cntup is
signal R : STD_LOGIC_VECTOR (3 downto 0):=(others => '0');
begin
process(clk)
begin
```

```
   if ( clk'event and clk ='1') then
      if (Reset = '1') then
         R <= (others => '0');
      elsif ce='1' then
         R<=R+1;
      end if;
   end if;
end process;
Q<=R;
end Behavioral;
--------------------
-- Registre LFSR
--------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity lfsr4 is
    Port ( clk : in  STD_LOGIC;
           ce : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0);
           Q1 : out  STD_LOGIC_VECTOR (1 to 4));
end lfsr4;
architecture Behavioral of lfsr4 is
signal reg:STD_LOGIC_VECTOR (4 downto 0):="00010";
signal d:STD_LOGIC_VECTOR (1 to 4):=(others => '0');
signal d0:std_logic:='0';
begin
process(clk)
begin
   if ( clk'event and clk ='1') then
      if (Reset = '1') then
         reg <= "00010";
      elsif ce='1' then
         reg(3 downto 1) <= reg(2 downto 0) ;
         reg(0) <= not(reg(4) XOR reg(3));
      end if;
   end if;
end process;
process(clk)
begin
```

```
   if ( clk'event and clk ='1') then
      if (Reset = '1') then
         d <= (others => '0');
      elsif ce='1' then
         d(2 to 4) <= d(1 to 3) ;
                  d(1)<=d0;
      end if;
   end if;
end process;

d0<=(d(4)xnor d(3))xor(d(1)and d(2) and d(3));
Q1<=d;
Q<=reg(3 downto 0);
end Behavioral;
----------------------------------------------
-- Démodulateur basé sur les registres LFSR
----------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity lfsrdemo is
    Port ( m : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           phi : out  STD_LOGIC_VECTOR (1 downto 0));
end lfsrdemo;
architecture Behavioral of lfsrdemo is
COMPONENT lfsrp
      PORT(
            D : IN std_logic_vector(3 downto 0);
            Reset : IN std_logic;
            clk : IN std_logic;
            ce : IN std_logic;
            Q : OUT std_logic_vector(3 downto 0)
            );
      END COMPONENT;
      COMPONENT lfsrn
      PORT(
            D : IN std_logic_vector(3 downto 0);
```

```
            Reset : IN std_logic;
            clk : IN std_logic;
            ce : IN std_logic;
            Q : OUT std_logic_vector(3 downto 0)
            );
      END COMPONENT;
signal m_1:std_logic:='0';
signal m_n:std_logic:='0';
signal RLP1:std_logic:='0';
signal RLP2:std_logic:='0';
signal RLN1:std_logic:='0';
signal RLN2:std_logic:='0';
signal Zero:std_logic_vector(3 downto 0):=(others=>'0');
signal Max:std_logic_vector(3 downto 0):="0001";
signal RP1,RP2,RN1,RN2:std_logic_vector(3 downto 0):="0001";
signal Qp1,Qn1,Qp2,Qn2:std_logic_vector(3 downto 0):="0001";
signal Cy:std_logic_vector(2 downto 0):="000";
signal R:std_logic_vector(7 downto 0):="00000000";
begin
sync:process(clk)
begin
      if rising_edge(clk) then
            if Reset='1' then
                  m_1<='0';
            else
                  m_1<=m;
            end if;
      end if;
end process;
m_n<=not m;
RLp1<= m_n and m_1;
RLn1<=m and not m_1;
RLp2<= RLn1;
RLn2<= RLp1;
R1R4:process(clk)
begin
      if rising_edge(clk) then
            if RLP1='1' then
                  RP1<=QP1;
```

```
                  RN2<=QN2;
            end if;
      end if;
end process;
R2R3:process(clk)
begin
      if rising_edge(clk) then
            if RLN1='1' then
                  RN1<=QN1;
                  RP2<=QP2;
            end if;
      end if;
end process;

lfsrp1: lfsrp PORT MAP(
            D => Zero,
            Reset => RLp1,
            clk => clk,
            ce => m,
            Q => Qp1
      );
lfsrn1:lfsrn PORT MAP(
            D => RP1,
            Reset => RLn1,
            clk => clk,
            ce =>m_n ,
            Q => Qn1
      );
lfsrp2:lfsrp PORT MAP(
            D => Zero,
            Reset => RLp2,
            clk => clk,
            ce => m_n,
            Q => Qp2
      );
lfsrn2:lfsrn PORT MAP(
            D => RP2,
            Reset => RLn2,
            clk => clk,
```

```
            ce => m,
            Q => Qn2
      );
process (clk)
begin
      if rising_edge(clk) then
            if Reset='1' or cy="101" then
                  cy<=(others=>'0');
                  R<=RN1&RN2;
            else
                  cy<=cy+1;
            end if;
      end if;
end process;
LUT00:with R select
phi<= "00" when X"00",


                  "00" when X"01",   "00" when X"19",   "00" when X"31",
"00" when X"49",    "00" when X"02",   "00" when X"1A",   "00" when X"32",
"00" when X"4A",    "00" when X"03",   "00" when X"1B",   "00" when X"33",
"00" when X"4B",    "00" when X"04",   "00" when X"1C",   "00" when X"34",
"00" when X"4C",    "00" when X"05",   "00" when X"1D",   "00" when X"35",
"00" when X"4D",    "00" when X"06",   "00" when X"1E",   "00" when X"36",
"00" when X"4E",    "00" when X"07",   "00" when X"1F",   "00" when X"37",
"00" when X"4F",    "00" when X"08",   "00" when X"20",   "00" when X"38",
"00" when X"50",    "00" when X"09",   "00" when X"21",   "00" when X"39",
"00" when X"51",    "00" when X"0A",   "00" when X"22",   "00" when X"3A",
"00" when X"52",    "00" when X"0B",   "00" when X"23",   "00" when X"3B",
"00" when X"53",    "00" when X"0C",   "00" when X"24",   "00" when X"3C",
"00" when X"54",    "00" when X"0D",   "00" when X"25",   "00" when X"3D",
"00" when X"55",    "01" when X"0E",   "00" when X"26",   "00" when X"3E",
"00" when X"56",    "00" when X"0F",   "00" when X"27",   "00" when X"3F",
"00" when X"57",    "00" when X"10",   "00" when X"28",   "00" when X"40",
"00" when X"58",    "00" when X"11",   "00" when X"29",   "00" when X"41",
"00" when X"59",    "00" when X"12",   "00" when X"2A",   "00" when X"42",
"00" when X"5A",    "00" when X"13",   "00" when X"2B",   "00" when X"43",
"00" when X"5B",    "00" when X"14",   "00" when X"2C",   "00" when X"44",
"00" when X"5C",    "00" when X"15",   "00" when X"2D",   "00" when X"45",
```

```
"00" when X"5D",    "00" when X"16",   "00" when X"2E",   "00" when X"46",
"00" when X"5E",    "00" when X"17",   "00" when X"2F",   "00" when X"47",
"00" when X"5F",    "00" when X"18",   "00" when X"30",   "00" when X"48",
"00" when X"60",    "00" when X"70",   "00" when X"97",   "00" when X"BE",
"00" when X"61",    "00" when X"71",   "00" when X"98",   "00" when X"BF",
"00" when X"62",    "00" when X"72",   "00" when X"99",   "00" when X"C0",
"00" when X"63",    "00" when X"73",   "00" when X"9A",   "00" when X"C1",
"00" when X"64",    "00" when X"74",   "00" when X"9B",   "00" when X"C2",
"00" when X"65",    "00" when X"75",   "00" when X"9C",   "00" when X"C3",
"00" when X"66",    "00" when X"76",   "00" when X"9D",   "00" when X"C4",
"00" when X"67",    "11" when X"77",   "00" when X"9E",   "00" when X"C5",
"00" when X"68",    "00" when X"78",   "00" when X"9F",   "00" when X"C6",
"00" when X"69",    "00" when X"79",   "00" when X"A0",   "00" when X"C7",
"00" when X"6A",    "00" when X"7A",   "00" when X"A1",   "00" when X"C8",
"00" when X"6B",    "00" when X"7B",   "00" when X"A2",   "00" when X"C9",
"00" when X"6C",    "00" when X"7C",   "00" when X"A3",   "00" when X"CA",
"00" when X"6D",    "00" when X"7D",   "00" when X"A4",   "00" when X"CB",
"00" when X"6E",    "00" when X"7E",   "00" when X"A5",   "00" when X"CC",
"00" when X"6F",    "10" when X"7F",   "00" when X"A6",   "00" when X"CD",
"00" when X"E5",    "00" when X"80",   "00" when X"A7",   "00" when X"CE",
"00" when X"E6",    "00" when X"81",   "00" when X"A8",   "00" when X"CF",
"00" when X"E7",    "00" when X"82",   "00" when X"A9",   "00" when X"D0",
"00" when X"E8",    "00" when X"83",   "00" when X"AA",   "00" when X"D1",
"00" when X"E9",    "00" when X"84",   "00" when X"AB",   "00" when X"D2",
"00" when X"EA",    "00" when X"85",   "00" when X"AC",   "00" when X"D3",
"00" when X"EB",    "00" when X"86",   "00" when X"AD",   "00" when X"D4",
                    "00" when X"87",   "00" when X"AE",   "00" when X"D5",
                    "00" when X"88",   "00" when X"AF",   "00" when X"D6",
                    "00" when X"89",   "00" when X"B0",   "00" when X"D7",
                    "00" when X"8A",   "00" when X"B1",   "00" when X"D8",
                    "00" when X"8B",   "00" when X"B2",   "00" when X"D9",
                    "00" when X"8C",   "00" when X"B3",   "00" when X"DA",
                    "00" when X"8D",   "00" when X"B4",   "00" when X"DB",
                    "00" when X"8E",   "00" when X"B5",   "00" when X"DC",
                    "00" when X"8F",   "00" when X"B6",   "00" when X"DD",
                    "00" when X"90",   "00" when X"B7",   "00" when X"DE",
                    "00" when X"91",   "00" when X"B8",   "00" when X"DF",
                    "00" when X"92",   "00" when X"B9",   "00" when X"E0",
                    "00" when X"93",   "00" when X"BA",   "00" when X"E1",
```

```
                    "00" when X"94",   "00" when X"BB",   "00" when X"E2",
                    "00" when X"95",   "00" when X"BC",   "00" when X"E3",
                    "00" when X"96",   "00" when X"BD",   "00" when X"E4",
                    "00" when X"EC",   "00" when X"F3",   "00" when X"FA",
                    "00" when X"ED",   "00" when X"F4",   "00" when X"FB",
                    "00" when X"EE",   "00" when X"F5",   "00" when X"FC",
                    "00" when X"EF",   "00" when X"F6",   "00" when X"FD",
                    "00" when X"F0",   "00" when X"F7",   "00" when X"FE",
                    "00" when X"F1",   "00" when X"F8",   "00" when X"FF",
                    "00" when X"F2",   "00" when X"F9",

            "00" when others;
end Behavioral;
```

Annexe C

Rapport de synthèse

C.1. SYNTHESE DE MODULATEUR

MODI Project Status			
Project File:	modi.ise	**Current State:**	Placed and Routed
Module Name:	top	**• Errors:**	
Target Device:	xc2v1000-4fg456	**• Warnings:**	
Product Version:	ISE 8.2i	**• Updated:**	mer. 19. sept. 22:11:55 2012

MODI Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	199	10,240	1%	
Number of 4 input LUTs	371	10,240	3%	
Logic Distribution				
Number of occupied Slices	199	5,120	3%	
Number of Slices containing only related logic	199	199	100%	
Number of Slices containing unrelated logic	0	199	0%	
Total Number 4 input LUTs	380	10,240	3%	
Number used as logic	371			
Number used as a route-thru	7			
Number used as Shift registers	2			
Number of bonded IOBs	20	324	6%	
IOB Flip Flops	8			
Number of GCLKs	1	16	6%	
Total equivalent gate count for design	4,859			
Additional JTAG gate count for IOBs	960			

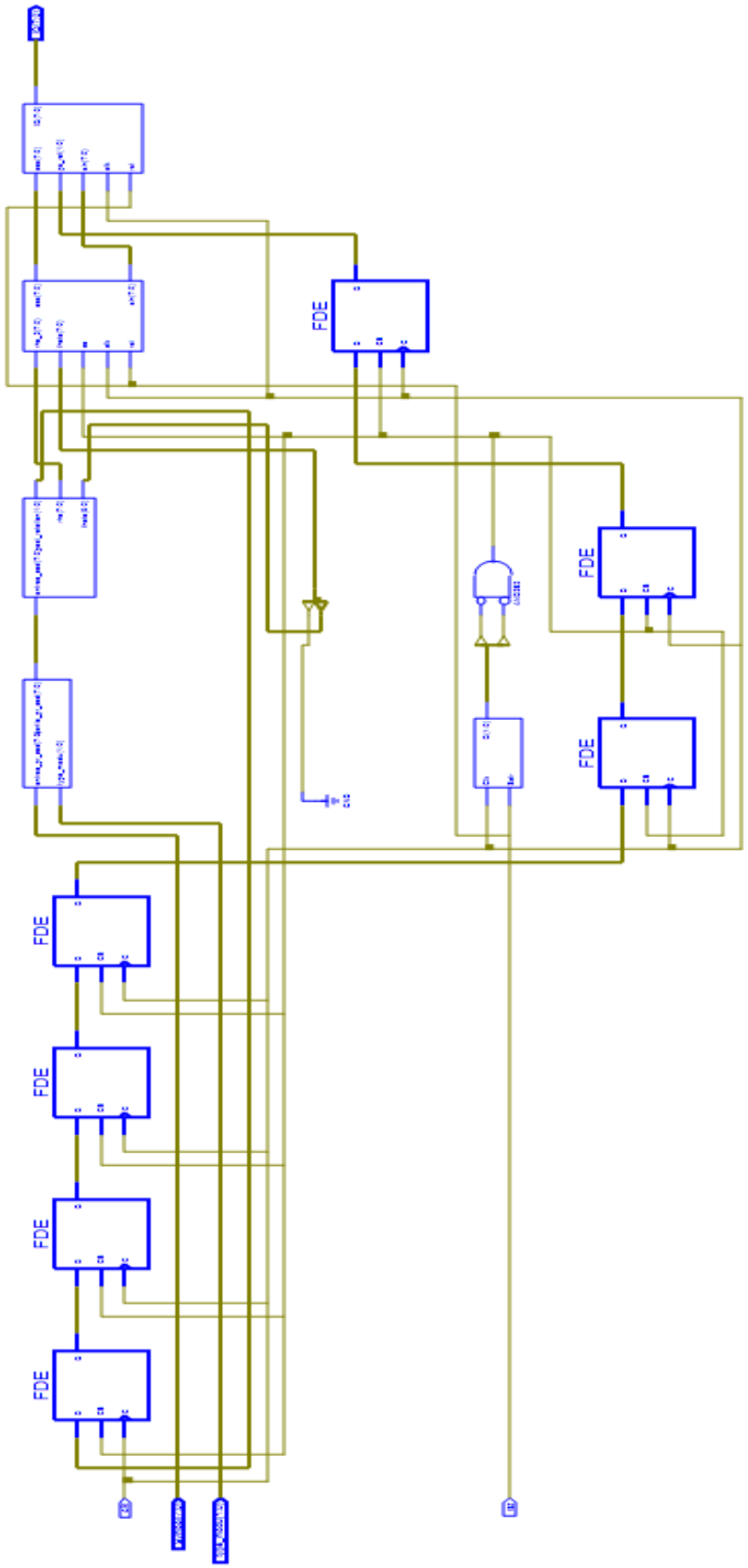
C.2. SYNTHESE DE DEMODULATEUR

LFSR Project Status			
Project File:	lfsr.ise	**Current State:**	Placed and Routed
Module Name:	lfsrdemo	**• Errors:**	No Errors
Target Device:	xc2v1000-4fg456	**• Warnings:**	5 Warnings (0 new)
Product Version:	ISE 8.2i	**• Updated:**	mer. 19. sept. 19:49:29 2012

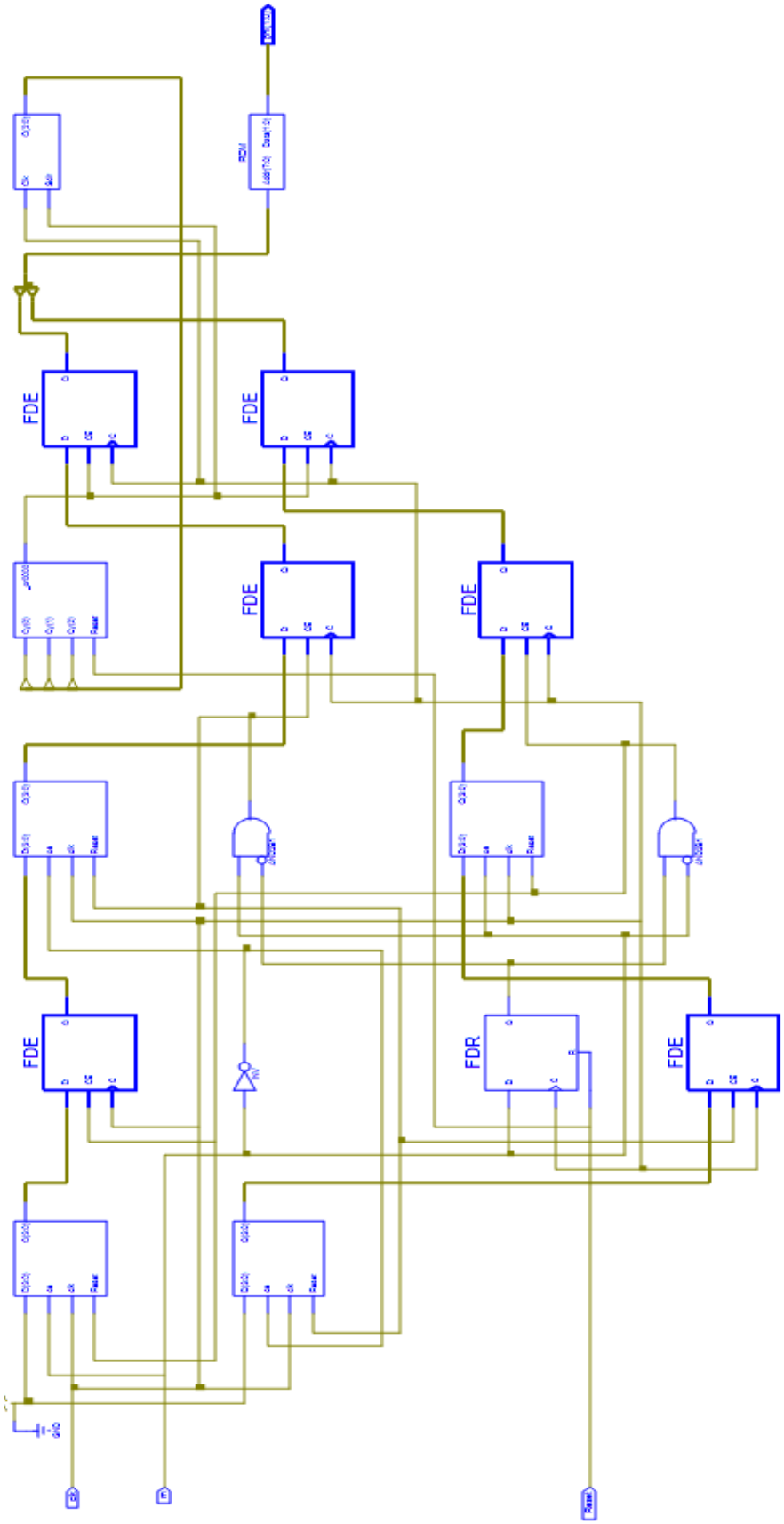
LFSR Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	43	10,240	1%	
Number of 4 input LUTs	32	10,240	1%	
Logic Distribution				
Number of occupied Slices	29	5,120	1%	
Number of Slices containing only related logic	29	29	100%	
Number of Slices containing unrelated logic	0	29	0%	
Total Number 4 input LUTs	32	10,240	1%	
Number of bonded IOBs	5	324	1%	
IOB Flip Flops	2			
Number of GCLKs	1	16	6%	
Total equivalent gate count for design	558			
Additional JTAG gate count for IOBs	240			

Annexe D



D.1. Schéma RTL interne du bloc modulateur



D.2. Schéma RTL interne du bloc démodulateur

Liste des Acronymes

ALU – Arithmetic Logic Unit

ASIC – Application Specific Integrated Circuit

ASK – Amplitude Shift Keying

AAL – Asynchronous Transfer Mode Adaptation Layer

ATM – Asynchronous Transfer Mode

BCH – Bose, Ray-Chaudhuri, Hocquenghem

CLB – Configurable Logic Block

CORDIC – COrdinate Rotation DIgital Computer

CPU – Central Processing Unit

dBc – Decibel Relative to Carrier

DSP – Digital Signal Processing

ETCD – Équipement Terminal de Circuit de Données

ETTD – Équipement Terminal Transmetteur de Données

FPGA – Field-Programmable Gate Array

FTP – File Transfer Protocol

FSK – Frequency Shift Keying

GF – Galois Field

IOB – Input Output Bloc

ISE – Integrated Software Environment

LFSR – Linear Feedback Shift Register

LUT – Look-Up Table

MAC – Medium Access Control

M-FSK – M-ary Frequency Shift Keying

M-PSK – M-ary Phase Shift Keying

OSI – Open System Interconnection

PLL – Phase-Locked Loop

PSK – Phase Shift Keying

QAM – Quadrature Amplitude Modulation

RAM – Random Access Memory

ROM – Read Only Memory

RTL – Register Transfer Level

TCP – Transmission Control Protocol

IP – Internet Protocol

UDP – User Datagram Protocol

UIT-T – Union Internationale des Télécommunications

VCO – Voltage Controled Oscillator

VHDL – Very High Speed Integrated Circuit Hardware Description Language