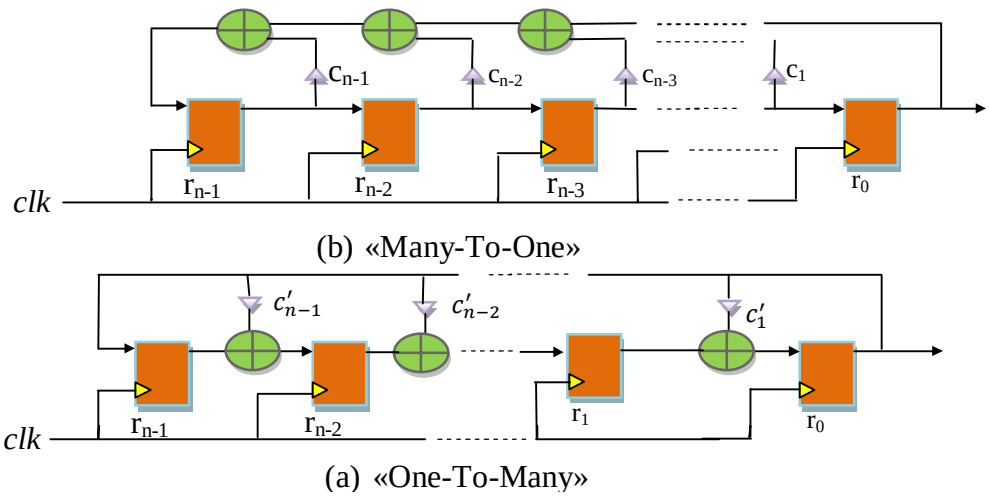


Figure IV.9. Les deux architectures de LFSR.

4.2. Le LFSR dans le codage de canal (code de convolution et turbo code)

Nous avons sans cesse besoin d'augmenter les débits de transmission tout en gardant ou en améliorant leur qualité. Malheureusement, différents types de bruits existent dans le canal de transmission ou dans les composants de la chaîne de transmission. Par conséquent, nous devons appliquer des techniques et envisager des stratégies [114] [116] pour assurer une transmission fiable des données:

- les stratégies ARQ (*Automatic Repeat Request*) qui sont basées sur la détection d'erreur et la retransmission; une demande envoyée à l'émetteur pour répéter le message sur la détection d'erreur par le récepteur,

- les stratégies Forward Error Correction (FEC) qui corrigent automatiquement les erreurs détectées au niveau du récepteur,

- Stratégies hybrides qui combinent ARQ et FEC.

Un code correcteur d'erreur (stratégies FEC) permet de corriger une ou plusieurs erreurs dans un mot de code en ajoutant aux informations des symboles redondants. Différents codes possibles existent mais dans notre travail, nous traitons seulement les codes Convolutifs et les Turbo code car ils présentent une similarité avec le code Gold et les démodulateurs numériques à base de registres LFSR. De plus ce sont les codes les plus utilisés dans les standards de télécommunications [11].

Le principe du codage convolutionnel a été introduit en 1955 par Elias comme alternative aux codes blocs [117]. Les codes convolutionnels ajoutent une certaine redondance aux bits de la séquence d'information à transmettre moyennant une opération logique (OU exclusif). Chaque information d'un symbole de m bits se transforme en un symbole de *n* bits, où m/*n* est le taux de code *(n ≥ m)*. Chaque codeur convolutionnel est caractérisé aussi par la longueur de contrainte K qui représente le nombre de cellules du registre à

décalage. Le code convolutif peut être divisé en deux catégories principales: le codeur non systématique (NSC) et le code convolutif récursif et systématique (RSC). Un code non systématique est celui dont la sortie ne contient pas les bits d'entrée. Dans sa forme la plus commune, un NSC est représenté par un *registre à décalage linéaire* et chaque code de sortie dépend d'un polynôme générateur dont l'équation pourrait être considérée comme l'équation d'un filtre de type FIR (Finite Impulse Response) [118]. Au contraire, un code systématique récursif est celui dans lequel la sortie contient les bits d'entrée plus une récurrence linéaire de la première sortie. Ainsi, une RSC est un codeur qui utilise un *registre à décalage à rétroaction linaire* (*LFSR*). Dans ce type de registre l'équation du polynôme générateur est une équation d'un filtre de type IIR (infinie Réponse Impluse):

$$y[n] = \sum_0^N b_k \,.\, x_{n-k} - \sum_1^N a_k . y_{n-k} \qquad (14)$$

où b_k et a_k sont respectivement les coefficients directs et de retour, N étant l'ordre du filtre.

Les codeurs convolutifs peuvent être mis en œuvre en utilisant l'une des deux structures équivalentes présentées dans la section suivante; le MTO et l'OTM.

4.2.1. Architecture MTO (Many To One).

La forme canonique d'un codeur convolutif MTO est représentée dans la figure IV.10 pour une mémoire du codeur de taille m = 3. où r_i représente la bascule i du registre R, g_i et h_i sont respectivement les coefficients du polynôme de la boucle de retour et du polynôme de la branche directe. Il y a deux configurations possibles :

- un codeur Convolutif non récursif (CNR) est obtenu, lorsque (g_0,g_1,g_2,g_3) = (0,*,*,*) or (1,0,0,0), * égal à 0 ou 1.

- un codeur convolutif récursif (CR) est obtenu, lorsque $g_0 = 1$ et (g_1, g_2, g_3) ≠ (0, 0, 0).

La boucle de rétroaction est inséré uniquement, lorsque $g_0 = 1$ et au moins un des coefficients g_1, g_2, g_3 est égal à 1.

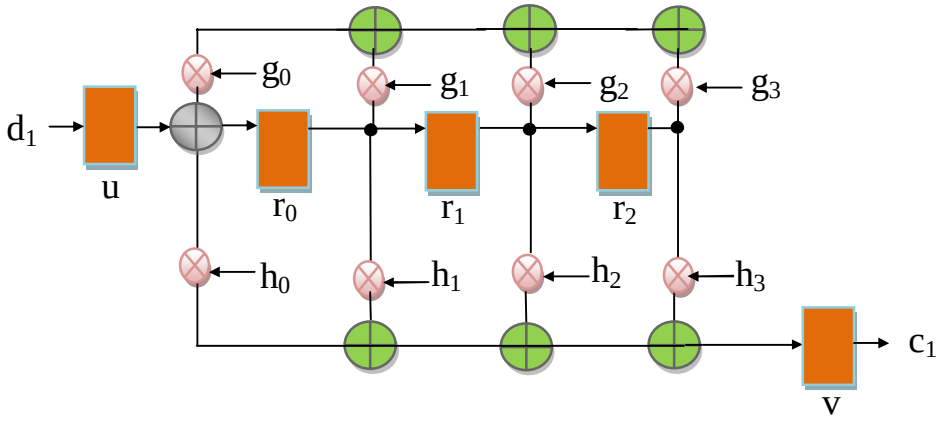


Figure IV.10. Codeur convolutif MTO pour m = 3.

4.2.2. Architecture OTM (One To Many)

La forme canonique d'un codeur convolutif série OTM est représenté dans la figure IV.11 pour M = 3. La configuration de la structure de l'encodeur OTM est similaire au MTO avec deux configurations possibles: Un codeur CNR est obtenu, lorsque (g_0,g_1,g_2,g_3) = (*,*,*,0) or (0,0,0,1). * égal à 0 ou 1, Un codeur convolutif récursif (CR) est obtenu, lorsque g_3 = 1 et (g_0, g_1, g_2) (0, 0, 0). La boucle de rétroaction est insérée uniquement lorsque g_3 = 1 et au moins l'un des coefficients g_0, g_1, g_2 est égal à 1.

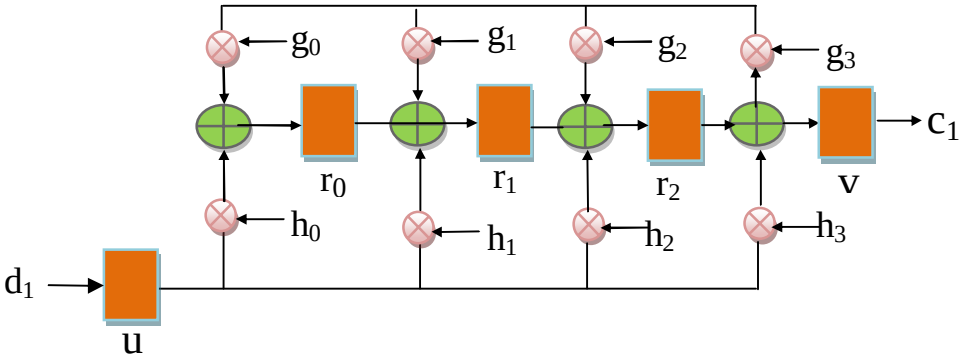


Figure IV.11. Codeur convolutif OTM pour m = 3.

4.2.3. Un turbo-code

Les turbo-codes peuvent être considérés comme une classe de codes concaténés. Un turbo-code est la concaténation d'un ou de plusieurs codes (convolutifs ou blocs) séparés par des blocs d'entrelacement. Suivant le type de la concaténation, nous avons les turbo-codes série et les turbo-codes parallèle.

Un turbo-code parallèle est construit par concaténation parallèle de deux codes de convolution séparés par entrelacement. Le premier code fonctionne avec la séquence d'informations et le second fonctionne avec la même séquence d'informations, mais entrelacées. La figure (IV.12) représente un turbo-code composé de deux codes constitutifs C_1 et C_2 de type RSC.

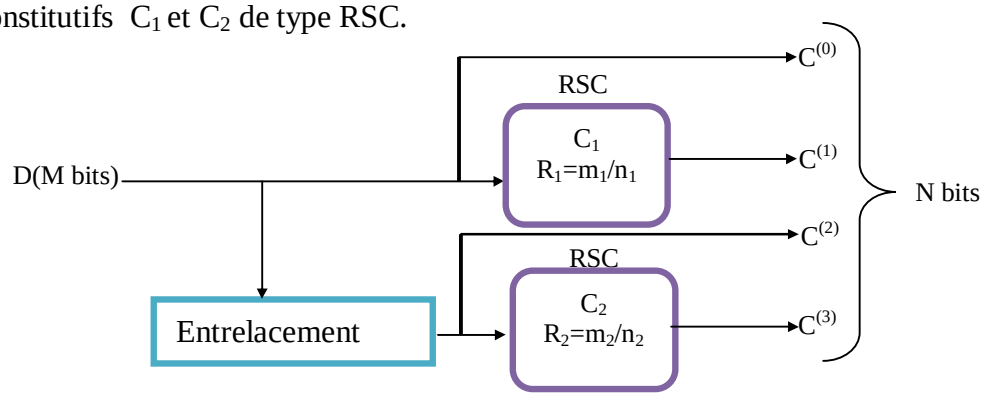


Figure. IV.12. Turbo-code parallèle.

Dans la pratique, nous choisissons les deux codes similaires et de structure RSC. En outre, afin d'augmenter la production, il est recommandé de ne pas utiliser la sortie systématique C $^{(2)}$.

4.3.Le LFSR pour le générateur de codes d'embrouillage dans le WCDMA

L'architecture du générateur de code Gold, basée sur une combinaison de deux registres à décalage à rétroaction linéaire (LFSR), est détaillée dans l'annexe A (section A.5.2).

4.4.LFSR pour la démodulation numérique

Les registres LFSR sont capables de produire des séquences fixes et périodiques, en fait c'est un choix pertinent pour une utilisation en tant que compteurs programmables [93] [109]. L'organisation fonctionnelle de l'un de ces compteurs (figure. IV.13) est composée de deux parties principales:

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

Le rôle de cette ROM est de permettre la conversion entre l'ordre de parcours des éléments de corps de Galois GF(2^n)et l'ordre «traditionnel» croissant utilisé dans les compteurs classiques. Une architecture, en respectant ces critères, se comporte comme un compteur modulo 2^n-1 [93][109].

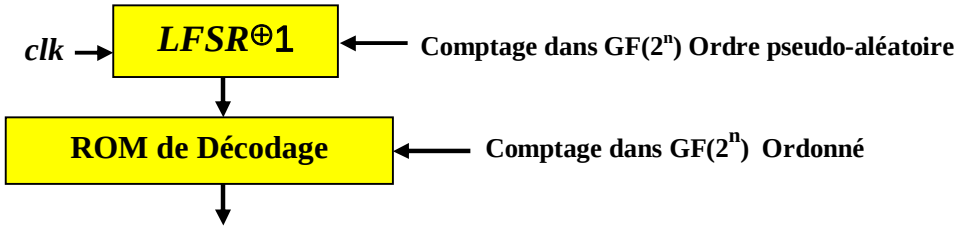


Figure. IV.13. Principe du compteur LFSR.

De nouvelles architectures [93][109] traduisent directement les fonctionnalités nécessaires à la démodulation des signaux en algorithmes adaptés aux circuits numériques et aux FPGA en particulier, ceci basant sur les registres LFSR. Les architectures des démodulateurs [93] utilisent ces compteurs pour mesurer directement la phase ou la fréquence des signaux qui lui sont fournies. L'architecture de quatre registres LFSR est conçue pour mesurer la phase dans le cas de la démodulation de phase, et celle de deux registres LFSR pour mesurer la fréquence dans le cas de la démodulation de fréquence [93][109].

4.5. Architecture générique proposée pour le codage et la démodulation numérique

A partir des architectures présentées précédemment pour la démodulation (phase et fréquence), pour le générateur de code de brouillage et pour le codage de type turbo code, nous pouvons remarquer que la principale similarité entre les quatre architectures est la

présence des registres LFSR [2]. Après avoir identifié ce point de similarité entre les architectures étudiées, nous essayons de proposer une architecture générique unique capable de prendre en charge les quatre fonctions tout en réduisant l'occupation de surface sur FPGA.

L'architecture universelle proposée (figure IV.14), est composée de quatre registres LFSR configurables, deux portes XOR, un bloc d'entrelacement, deux LUT (ROM), un bloc pour le calcul de la différence, un registre de données et un multiplexeur de configuration [2].

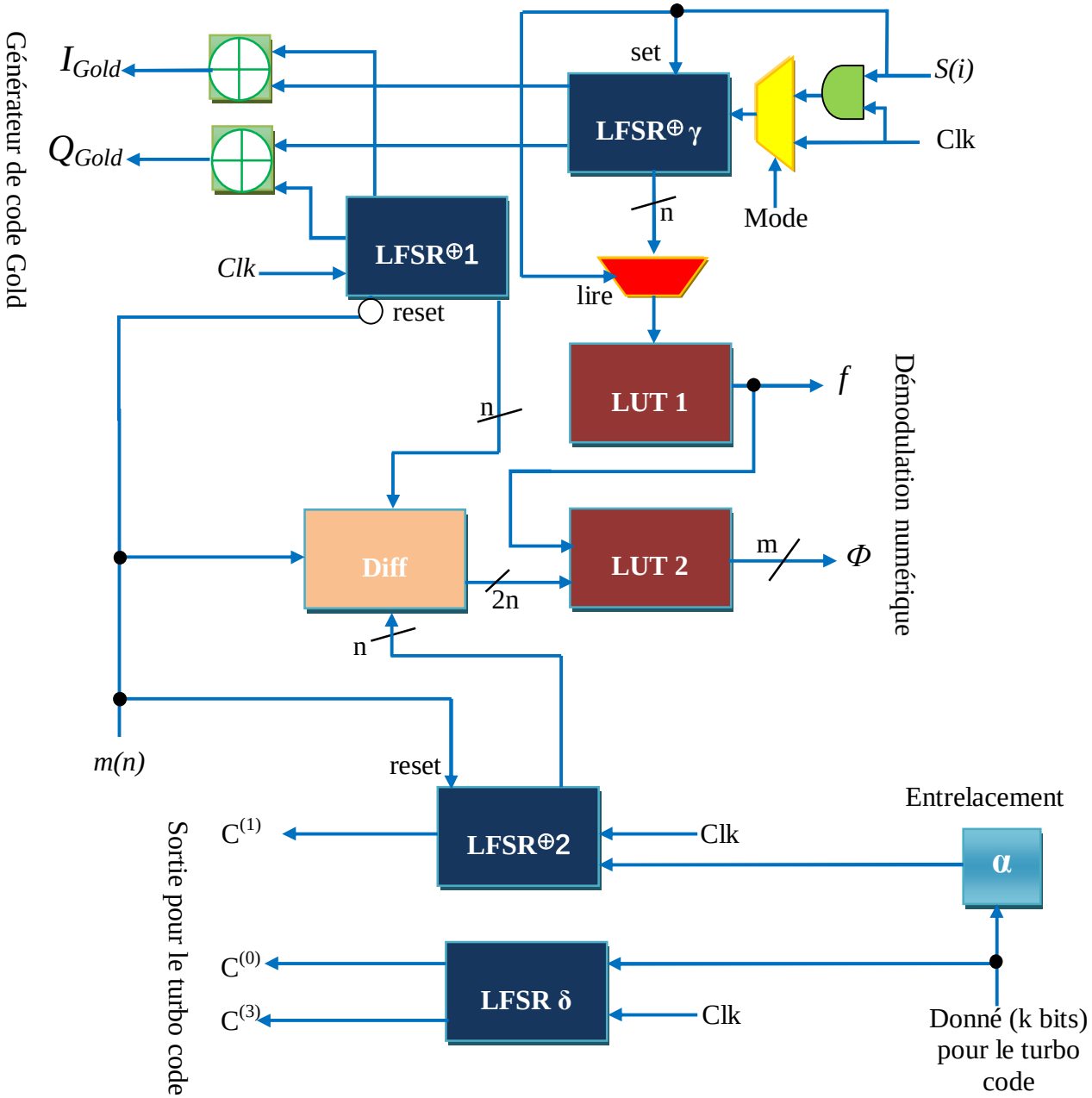


Figure. IV.14. Architecture générique proposée [2].

Dans cette architecture, le signal *mode* (figure IV.14) est un paramètre booléen (mis en œuvre par le biais d'un de bit de sélection), il permet de configurer l'architecture en utilisant un multiplexeur de configuration (couleur jaune dans la figure IV.14).

Dans la section suivante, nous proposons une structure d'un registre LFSR reconfigurable pour l'architecture générique (figure IV.14). Ce registre reconfigurable peut être considéré comme une architecture commune pour la démodulation numérique et le codage.

4.5.1. Un registre LFSR reconfigurable comme un opérateur commun

L'architecture du registre LFSR utilisée se diffère en fonction de l'application et de la fonctionnalité, par exemple le générateur de code Gold utilise deux registres où chaque registre possède une largeur (taille) de 18 bits et les deux se diffèrent dans les polynômes générateurs. La longueur du registre utilisée dans le cas de code Gold n'est pas forcément la longueur qui doit être utilisée pour les LFSR du codeur convolutif ou le démodulateur numérique. En effet, cette différence dans l'architecture de LFSR peut se résumer comme suit:

- le nombre de cellules du registre à décalage ou la longueur de ce registre,
- la différence dans le degré du polynôme générateur,
- la différence dans les polynômes générateurs (les coefficients g_i et h_i),

Par conséquent, nous avons proposé un registre (figure IV.15), reconfigurable considéré comme un opérateur commun pour plusieurs fonctions [2]. Nous remarquons que les registres dans notre architecture proposée (figure IV.14) se retrouvent dans l'architecture du LFSR reconfigurable (figure IV.15) [2].

L'architecture proposée (figure IV.15) se compose de *"n"* flip-flop D, des portes logique, des coefficients (g_i/ h_i) et des paramètres booléen "S_i " (mis en œuvre par le biais d'un bit de sélection). Ce registre peut s'adapter facilement à la nouvelle application par une simple configuration de ces paramètres "S_i " et ces coefficients (g_i/ h_i) [2]. L'entrée d'horloge de chaque bascule « r_i » est le résultat de la multiplication du signal *CLK* (l'horloge de FPGA) et un paramètre booléen "S_i ", où *i* représente l'ordre de la bascule considérée dans le registre. Donc si "S_i" est égal à '0' la bascule « r_i » ne fonctionne plus (désactivée), et si "S_i " est égal à '1' la bascule « r_i » est active (elle fonctionne). Ceci permet de changer le degré du polynôme du registre LFSR ainsi que leur structure interne. De plus, les coefficients (g_i/h_i) dans le registre universel sont modifiables et reconfigurables en fonction de l'application. Le changement des coefficients (g_i/h_i) permet de changer le polynôme générateur de ce registre. De cette façon, notre registre proposé peut être adapté au changement de la fonction par la paramétrisation des coefficients (g_i/h_i) et les paramètres "S_i".

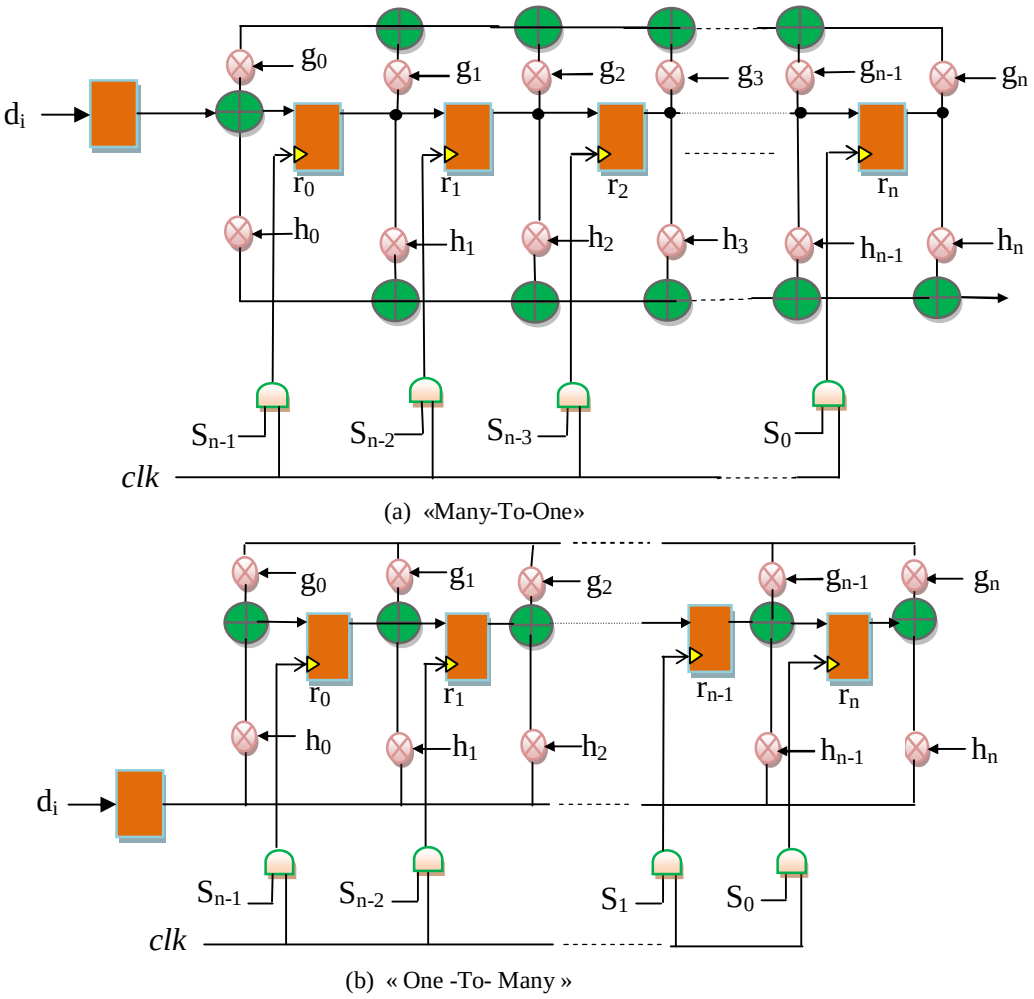


Figure. IV.15. Les deux architectures proposées LFSR configurables [2].

La configuration de ce registre reconfigurable est faite par un autre registre externe (registre paramètre de reconfiguration dans la figure IV.16) qui contient les valeurs des paramètres "S_i" et les coefficients (g_i/h_i) (figure IV.16).

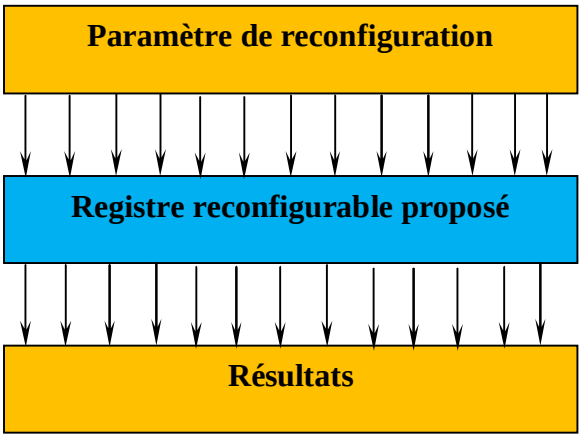


Figure. IV.16. Technique de reconfiguration du registre LFSR proposé.

Ainsi, la sortie de chaque bascule de ce registre reconfigurable est reliée avec un autre registre 'Résultats' (figure IV.16). Ce registre (Résultats) contient le résultat du registre reconfigurable proposé.

4.5.2. Fonctionnement de l'architecture flexible

L'architecture proposée (figure IV.14) peut basculer entre deux modes qui sont le mode démodulation numérique et le mode codage. L'architecture peut prendre en charge l'un des deux modes, mais pas les deux modes en même temps; c'est-à-dire lorsque l'architecture est en mode démodulation, elle ne peut pas fonctionner en mode codage et vise versa.

4.5.2.1. Mode démodulation numérique

Dans ce mode, l'architecture peut fonctionner comme un démodulateur de phase et un démodulateur de fréquence.

- **Mode: démodulation de fréquence FSK**

Dans ce mode, le registre (figure IV.14) nommé LFSR⊕γ est utilisé pour la démodulation de fréquence de ce signal. La première étape est la conversion du signal échantillonné et quantifié en un signal purement binaire. Cette première étape est indispensable avant de commencer la démodulation, puisque les LFSR fonctionnent avec des signaux binaires seulement. Il faut noter aussi que ce signal converti doit présenter la même fréquence et la même phase que le signal original, dont l'amplitude ne peut prendre d'autres valeurs que 0 ou 1 (voir annexe C). Le signal à démoduler et converti en forme binaire est noté $S(i)$. Ce signal représente une des deux entrées d'un multiplexeur (couleur jaune dans la figure IV.14). Ce multiplexeur a une sortie et deux entrées: l'horloge (*Clk*) et le signal $S(i)$ multiplié par l'horloge ($S(i) \times Clk$). Le signal de contrôle de ce multiplexeur est le signal *mode*. Ceci permet de sélectionner selon le mode de fonctionnement soit le signal à démoduler en fréquence $S(i)$ multiplié par l'horloge (*clk*) soit seulement l'horloge (*Clk*) [2]. Par exemple, dans ce mode le signal *mode* permet pour le signal ($S(i) \times Clk$) de passer dans le registre *LFSR⊕γ*.

Le polynôme générateur du registre *LFSR⊕γ* est de degré *n*, il pourra passer par (2^n-1) états différents. Le signal de comptage qui anime le *LFSR⊕γ* est le signal à démoduler ($S(i) \times Clk$), donc le *LFSR⊕γ* est incrémenté sur les fronts montants du signal ($S(i) \times Clk$). De plus, on a utilisé le signal à démoduler $S(i)$ comme un signal « reset » pour ce registre (*LFSR⊕γ*).

Si un front descendant (l'état logique «0») est détecté du signal $S(i)$, le contenu de ce registre est lu par un autre registre de données (couleur rouge dans la figure IV.14) juste à l'instant du déclenchement de ce signal « reset ». Le registre de données (couleur rouge

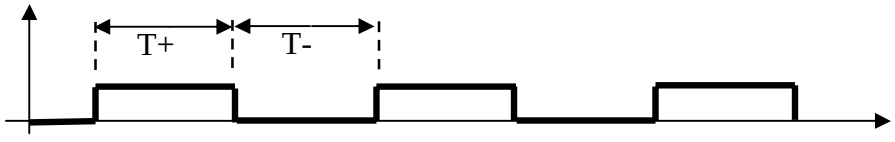
dans la figure IV.14), stocke la valeur de l'état interne du registre LFSR⊕γ à chaque incrémentation. En même temps, le registre LFSR⊕γ est remis à sa valeur initiale par ce signal *reset* (*S*(*i*)).

Enfin, la valeur lue est utilisée pour adresser une *LUT* (LUT 1) qui contient une table de correspondance entre l'ensemble des valeurs possibles pour le LFSR⊕γ et la fréquence associée (f)[2][109]. Cette fréquence mesurée, représente le symbole qui a été transmis par la partie modulation.

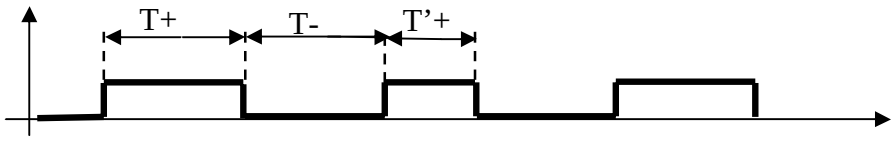
- **Mode: Démodulation de phase PSK**

Dans ce mode, les deux registres notés *LFSR*⊕1 et *LFSR*⊕2, fonctionnant en mode compteur peuvent mesurer la phase de ce signal. D'abord, comme pour la modulation de fréquence, le signal d'entrée est converti en une forme purement binaire, mais conserve la même fréquence et la même phase que le signal original (voir annexe C). Le signal M-PSK après une conversion en binaire est noté *m*(*n*) (figure IV.14). Le principe de la démodulation de phase est basé sur la mesure de la *régularité* du signal *m*(*n*) [93]. En d'autres termes, un changement de phase sur un signal binaire carré se traduit par une rupture de la régularité des périodes pendant lesquelles le signal est à l'état haut et à l'état bas. Dans le cas d'un signal ne subissant aucun déphasage, la durée de ces périodes doit être identique (figure IV.17) [93]. Sur le chronogramme (IV.17a), $T^+ = T^-$ le signal conserve la même phase sur toute la durée du graphe. Par contre, sur la figure (IV.17b), $T'_+ < T^-$ ce qui dénote un déphasage $\Delta\theta$. Celui-ci peut-être évalué de la façon suivante [93]:

$$\Delta\theta = 2\pi f(T'_+ - T^-) \qquad (15)$$



(a) Signal non déphasé



(b) Signal déphasé

Fig. IV.17: Durées des états haut et bas pour un signal non déphasé, et un signal avec un déphasage.

Par conséquent, on a utilisé deux registres: le registre LFSR⊕1 permet de mesurer la durée pendant laquelle le signal *m*(*n*) reste à l'état haut et le LFSR⊕2 permet de mesurer la durée pendant laquelle le signal *m*(*n*) reste à l'état bas [2]. Les deux registres (*LFSR*⊕1 et *LFSR*⊕2) fonctionnent comme deux compteurs synchrones incrémentés sur le front montant de l'horloge *clk*. Pour mesurer la durée de l'état haut et l'état bas, on a utilisé le signal à démoduler *m(n)* comme un signal « reset » pour les deux registres de cette façon:

- si un front montant du signal *m(n)* (état logique «1») est détecté, le LFSR⊕1 commence l'incrémentation, après détection d'un front descendant de *m*(*n*) le contenu de ce registre est lu par le bloc « *Diff* », et ce registre est remis à sa valeur initiale par le signal reset *m(n)* (figure IV.14).

- si un front descendant du signal *m(n)* (état logique «0») est détecté, le LFSR⊕2 commence l'incrémentation, après détection d'un front montant de *m*(*n*) le contenu de ce registre est lu par le bloc « *Diff* », et ce registre est remis à sa valeur initiale par le signal reset *m(n)* (figure IV.14).

Le module « *Diff* », permet de calculer la différence des résultats des deux compteurs pour le calcul de déphasage du signal *m*(n). Ce déphasage est calculé à l'aide de l'équation (15), et la conversion finale est effectuée par une LUT (LUT 2) contenant la table de conversion entre les couples de valeurs (R_i, f) et la phase θ. Pour la fréquence qui entre dans l'équation (15), nous supposons qu'elle est évaluée en utilisant une autre architecture de démodulation de fréquence comme celle présentée dans la section suivante. Enfin, la phase mesurée représente le symbole qui a été transmis par la partie modulation.

Nous pouvons utiliser deux LUTs qui contiennent les valeurs des symboles correspondants, par exemple la première LUT est adressée par la valeur de la fréquence et la deuxième LUT est adressée par la valeur de la phase.

4.5.2.2. Mode codage

Dans ce mode l'architecture peut fonctionner comme un générateur de code Gold et un turbo code de type parallèle.

- **Mode générateur de code de Gold**

Le générateur de code Gold nécessite deux registres LFSR (section 3.3), donc nous avons utilisé deux registres configurables nommée *LFSR*⊕γ et *LFSR*⊕1. Ainsi, nous devons adapter les coefficients h_i, g_i et les paramètres S_i pour la génération de code de Gold, car les deux registres ont une longueur de 18 bits, et les deux polynômes utilisés (deux séquences m) sont décrits par les équations A.12 et A.13 respectivement (voir annexe A). Donc, le premier LFSR noté *LFSR*⊕γ est de polynôme générateur: $1 + X^7 + X^{18}$, et le

deuxième LFSR noté *LFSR*⊕1 est de polynôme générateur: $1 + X^5 + X^7 + X^{10} + X^{18}$. Le signal de comptage qui anime les deux registres (*LFSR*⊕γ et *LFSR*⊕1) est l'horloge (*CLK*). Par conséquent, le signal de commande *Mode* permet de sélectionner le signal *CLK* comme une sortie du multiplexeur (couleur jaune dans la figure IV.14). Enfin, nous avons besoin de deux portes XOR (figure IV.14) pour la génération de la partie réelle et imaginaire du code Gold [2][9].

- **Mode codage de canal.**

L'architecture universelle proposée peut prendre en charge la fonction d'un turbo code de type parallèle en utilisant deux registres LFSR noté *LFSR*⊕δ et *LFSR*⊕2 (figure IV.14). À ce propos, l'architecture des deux registres doit être modifiée conformément aux architectures d'un codeur convolutif récursif et systématique (RSC) (section 4.2). Ce changement est facilement obtenu car tous les LFSR dans l'architecture proposée sont des registres reconfigurables (figure IV.15).

Nous pouvons donc changer le degré du polynôme du registre (ou la taille de la mémoire du codeur), soit par exemple les paramètres S_i comme (S_{n-1} S_{n-2} S_{n-3}…….. $S_{0)}$ = (11100……..0) pour obtenir un codeur convolutif de taille (de mémoire) m = 3. Les coefficients (g_i) et (h_i) dans le registre configurable sont modifiables et reconfigurables en fonction de l'application [2]. En plus, l'entrée pour le deuxième registre est entrelacée par le bloc d'entrelacement.

5. Implémentation sur FPGA.

La description RTL des architectures proposées dans ce chapitre a été implémentée en utilisant la description structurale VHDL (VHSIC Hardware Description Language). Le FPGA ciblé pour implémentation est la famille Virtex-V de Xilinx [94]. L'outil ISE 10.1i de Xilinx [51] a été utilisé pour cette implémentation numérique permettant d'obtenir les ressources logiques nécessaires et les performances associées.

5.1. Implémentation de l'architecture générique proposée pour la modulation numérique

Dans cette section, nous présentons les résultats de la synthèse et la simulation de l'architecture numérique générique pour la modulation M-QAM, M-PSK et B-FSK. L'architecture matérielle proposée de cette architecture générique est écrite en langage VHDL, en utilisant l'outil de conception Xilinx ISE 10.1i pour la synthèse. La conception RTL pour le codeur, la LUT de phase et la LUT d'amplitude, le processor Cordic et l'additionneur pipeline sont montrés dans les figures IV18. IV.19, IV.20, IV.21 et IV.22 respectivement. Le tableau IV.1 montre les résultats de synthèse de l'ensemble du système sur des données de 8 bits.

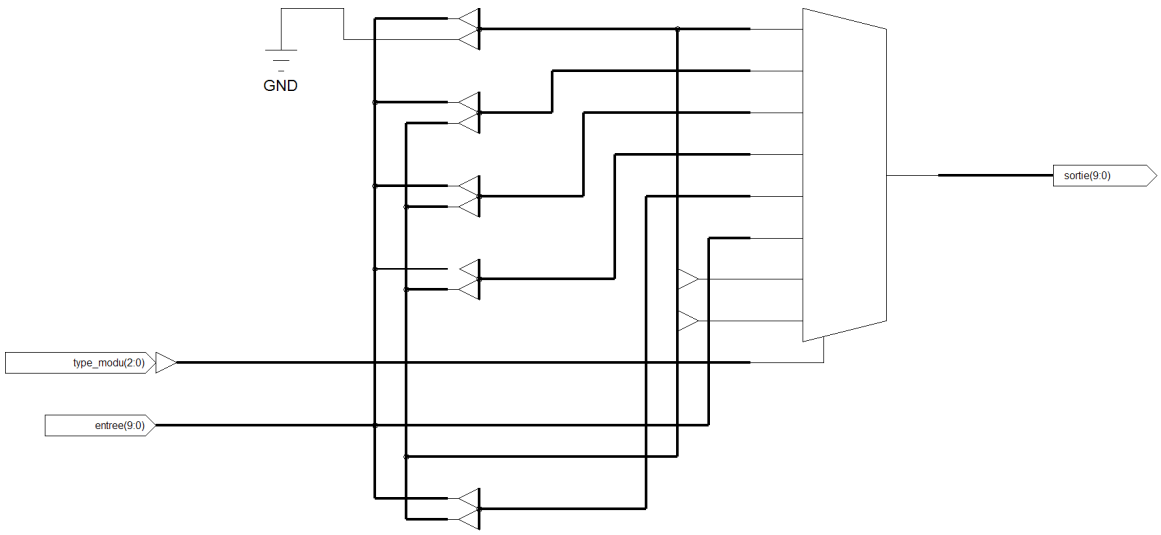


Figure IV.18. Schémas RTL de l'implémentation sur FPGA Virtex 5 du codeur.

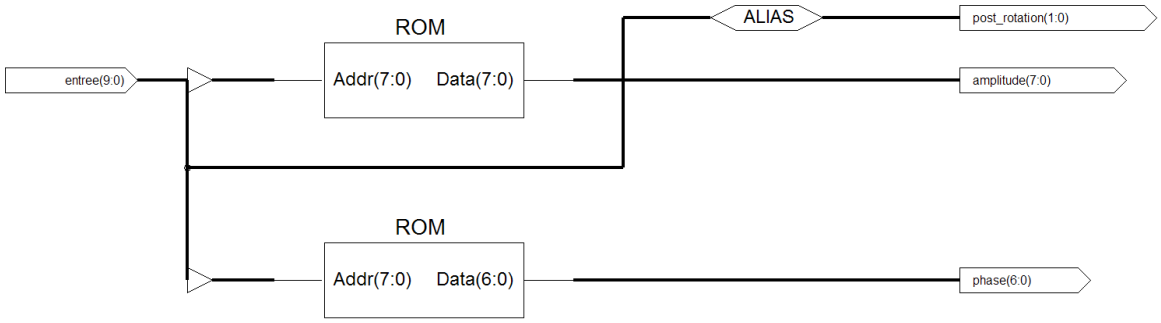


Figure IV.19. Schémas RTL de l'implémentation sur FPGA Virtex 5 de LUT de phase et la LUT d'amplitude.

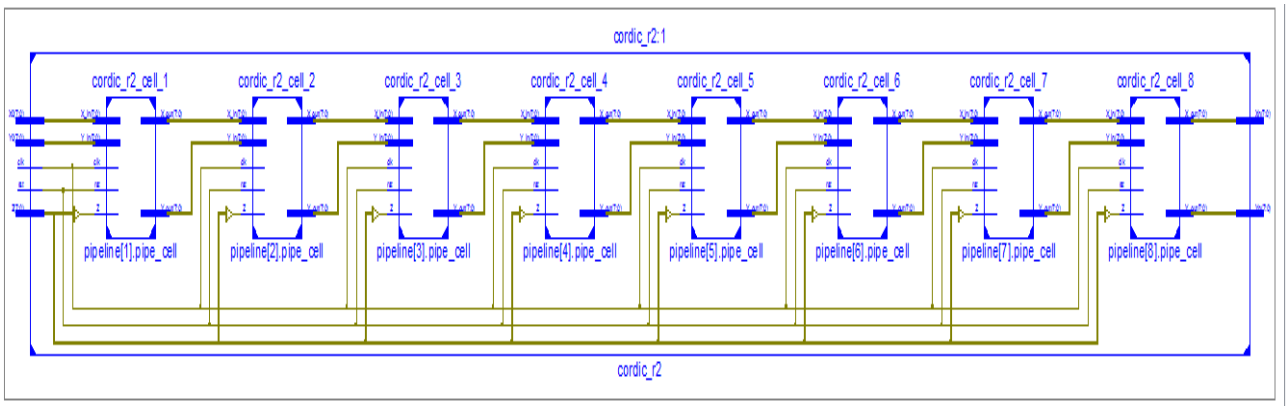


Figure IV.20. Schémas RTL de l'implémentation sur FPGA Virtex V du CORDIC de 8 étages.

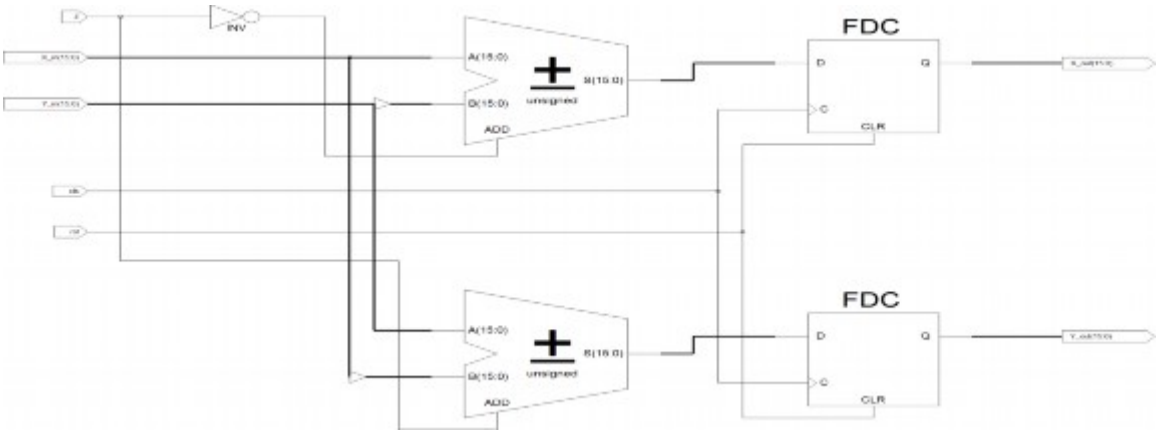


Figure IV.21. Schémas RTL d'un étage d'algorithme de CORDIC proposé par [93].

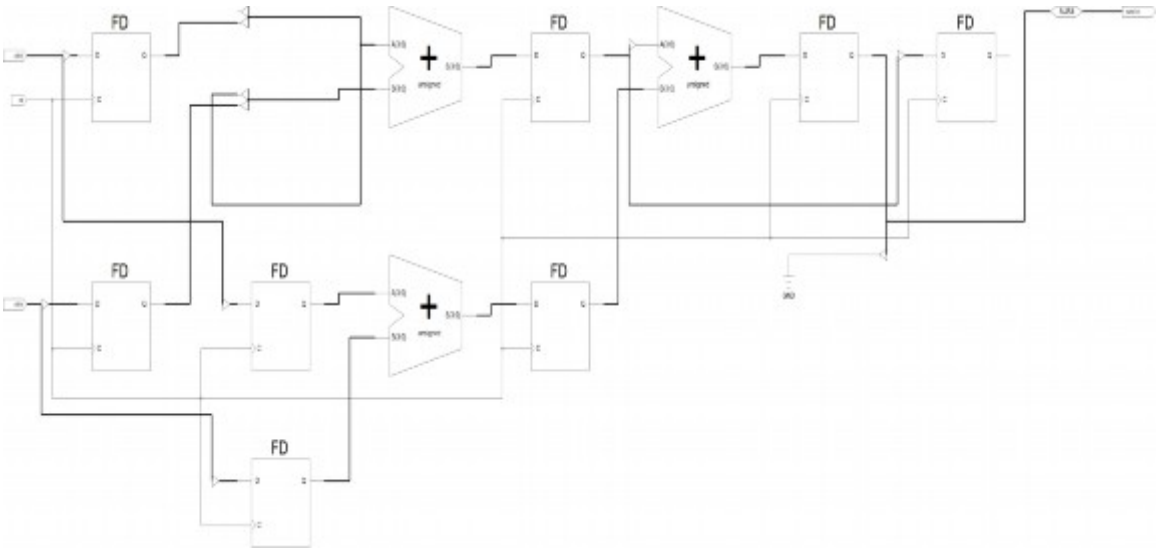


Figure IV.22. Schémas RTL de l'additionneur pipeline.

Tableau IV.1. Résultats de synthèse de l'architecture proposée

	Slices registres (19200)	LUT (19200)	Fréquence de fonctionnement (MHz)
Codeur	----	12	------
LUT de phase et LUT d'amplitude	----	15	------
CORDIC	128	136	602.518
Additionneur pipeline	29	13	645.953

Nous pouvons déduire de ce tableau que l'architecture proposée possède une faible consommation de ressource avec des donnés de 8 bits. En effet, chaque bloc de cette architecture utilise un très faible pourcentage (%) des éléments logiques disponibles dans le circuit ciblé, conduisant à un circuit de très petite dimension. On peut voir même avec le processeur Cordic, que nous avons une faible consommation, ceci revient à l'utilisation du Cordic optimisé proposé par [93]. Cette très faible consommation en ressource de cette architecture donne la possibilité de faire la parallélisation pour augmenter la fréquence de fonctionnement et la fréquence d'échantillonnage apparente ou d'implanter d'autres architectures en utilisant le reste des ressources.

- **Résultats de simulation**

Pour la vérification de nos implémentations, nous avons utilisé ISE 10 pour la simulation des résultats. Le but de cette simulation consiste à vérifier le fonctionnement du bloc additionneur pipeline proposé et le processeur Cordic car leur vitesse de fonctionnement est déterminante dans l'architecture proposée. Un fichier testbench a été écrit spécialement pour cette simulation. Les figures IV.23 et IV.24 montrent les résultats de simulation de l'implémentation proposée.

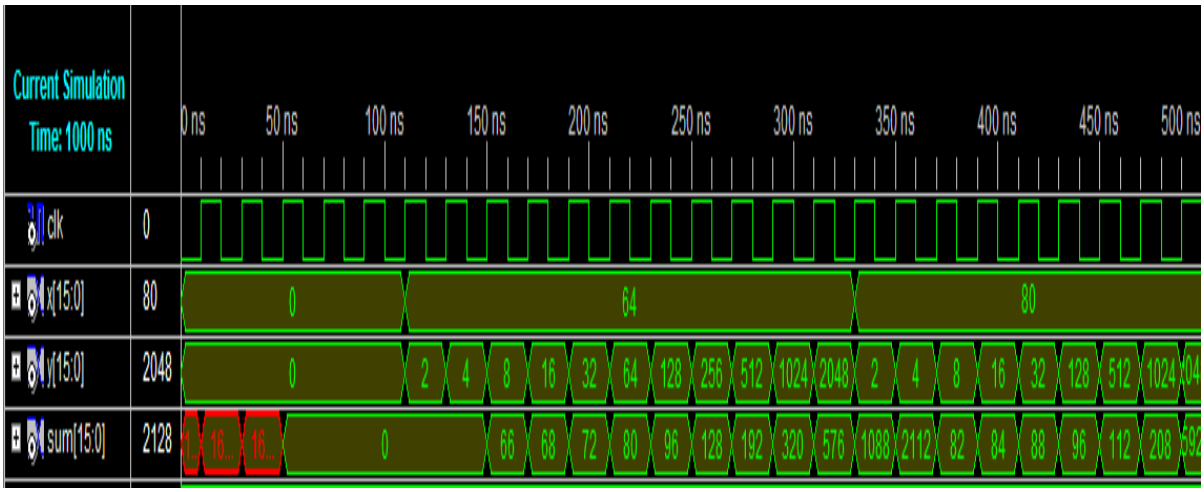


Figure IV.23. Résultats de simulation de l'additionneur pipeline.

Sur le chronogramme de la figure IV.23,

- Le signal «*X*» *représente la phase issue de la LUT de phase*,
- Le signal «*Y*» sont les N sorties de l'accumulateur de phase, elles assurent la description discrète du signal sinusoïdal désiré,
- Le signal «sum» représente la sortie de l'additionneur pipeline.

On remarque que le résultat de l'addition est disponible après 2 cycles de retards, ce qui est clair car nous avons dans le premier cycle une opération d'addition des MSB pour les

deux signaux d'entrée et une opération d'addition des LSB pour les deux signaux d'entrée, puis dans le deuxième cycle nous avons l'opération finale (figure IV.6).

Pour le Cordic sur le chronogramme de la figure IV.24 on a :

- le signal « *X* » représente l'amplitude du signal à générer,
- le signal « *Z* » » représente la phase du signal à générer,
- le signal « Xn et Yn » représentent la sortie de Cordic, où exactement le signal « Xn » représente le cosinus de la phase et « Yn » représente le sinus de la phase.

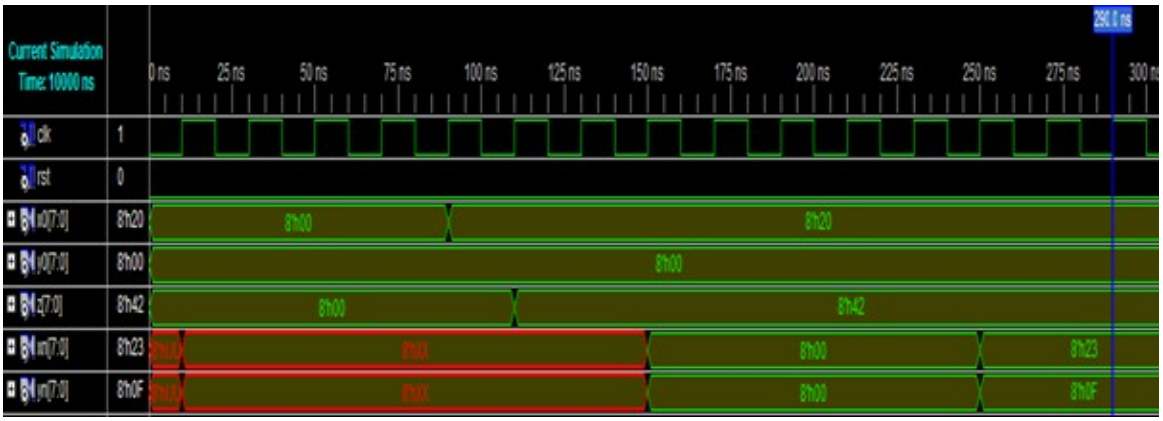


Figure IV.24. Résultats de simulation du Cordic.

Sur le chronogramme de la figure IV.24 nous avons montré le calcul d'un seul angle d'un seul échantillon (de valeur 42 en hexa), on remarque que le résultat est disponible après 8 cycles d'horloge. Ceci s'explique par le fait que nous avons utilisé un Cordic de 8 étages. Ce résultat montre bien l'intérêt de paralléliser plusieurs processeurs Cordic afin de générer plusieurs échantillons d'une même sinusoïde en parallèle. On peut par exemple paralléliser 8 processeurs Cordic pour obtenir 8 échantillons en 8 cycles d'horloge au lieu d'un seul échantillon en 8 cycles d'horloge.

5.2. Implémentation de l'architecture générique pour le codage et la démodulation numérique.

Dans cette section, nous présentons les résultats de la synthèse et la simulation de l'architecture universelle proposée pour le codage canal, le code de brouillage et la démodulation numérique. De plus, pour une évaluation réelle de la performance et la consommation, nous comparons les implémentations de notre architecture générique avec la technique Velcro.

L'architecture de Velcro utilisée pour la comparaison est composée de quatre architectures différentes :

- les architectures de démodulation de phase [93] de quatre registres,

- les architectures de démodulation de la fréquence [93][109] de deux registres,
- le générateur de code de Gold avec deux registres,
- le Turbo code avec deux registres.

Toutes les architectures sont codées en VHDL, simulées et synthétisées en utilisant le ModelSim SE v6.5 et l'outil Xilinx ISE 10. La figure IV.25 présente le schéma RTL pour notre architecture proposée.

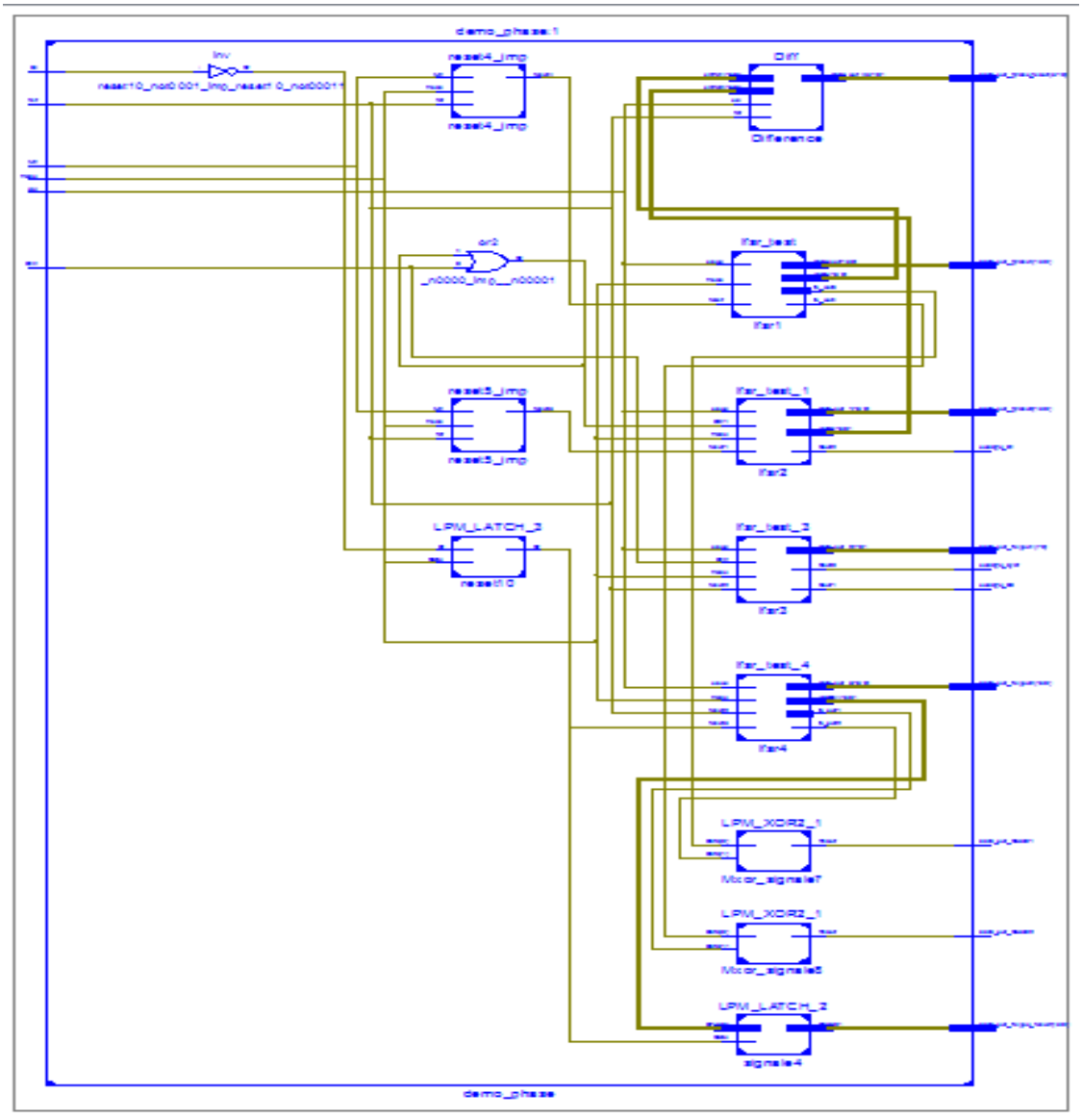


Figure. IV.25. Schémas RTL de l'implémentation sur FPGA Virtex V de l'architecture proposée [2].

Les résultats de l'implémentation après la synthèse par l'outil ISE.10 de Xilinx sont donnés dans le tableau IV.4 pour différentes tailles (longueur de LFSR) en termes de

surface occupée (le nombre de Slice LUTs (Look Up Tables) et le nombre de slice registres), la fréquence de fonctionnement maximal et la consommation de puissance.

Les polynômes générateurs pour les codes Convolutifs utilisés pour cette simulation sont donnés dans le tableau IV.2 et le tableau IV.3. Leur choix est justifié par le fait qu'ils présentent la meilleure distance libre pour une taille de mémoire donnée [118].

Notations: Chaque codeur est défini par la notation (k, n, m) $(G, H_1, \ldots, H_i)_8$; où:

- k représente le nombre d'entrées du codeur,
- n représente le nombre de sorties,
- M est la taille de la mémoire du codeur,
- G représente les coefficients du polynôme de la boucle récursive,
- $H_1, \ldots\ldots, H_i$ sont les i polynômes correspondant aux i sorties codées,
- $()_8$ signifie que les polynômes générateurs sont donnés en octal.

Tableau. IV.2. Polynômes générateurs de codes Convolutifs.

R=1/2
(1,2,2)(7,5)8
(1,2,4)(23,33)8

Tableau. IV.3. Polynômes générateurs de codes Convolutifs (R=1/2)

R=1/2		
Notation	Degré	Pôlynomes de la boucle récursif
G(X)	16	$1 + X^2 + X^{15} + X^{16}$
Notation	Degré	Pôlynomes de la branche directe
H(X)	16	$1 + X^1 + X^2 + X^4 + X^7 + X^9 + X^{10} + X^{14} + X^{16}$

Dans le tableau IV.4, on observe que la performance de l'architecture proposée est meilleure que celle de l'architecture Velcro. L'occupation de surface est réduite de 40% à 50% comparée à l'architecture Velcro. Notre architecture proposée pour 4 bits prend seulement 65 « slice registers » et atteint une fréquence de 644 MHz, en fonctionnement maximum avec la technologie Virtex 5.

Notre architecture comparée à l'architecture Velcro, offre un gain de LUT entre 11,32% et 35,07%. La consommation de surface de notre architecture progresse linéairement avec la précision en bit, elle présente toujours une consommation raisonnable qui permet d'envisager une parallélisation éventuelle en multipliant les étages de calcul. Nous

pouvons considérer aussi qu'il reste suffisamment d'espace afin d'implémenter des algorithmes additionnels.

Tableau. IV.4. Comparaison de l'architecture proposée avec l'architecture Velcro.

Xilinx Virtex 5						
	Résolution					
	Longueur du LFSR (bits) de 4 bits pour la démodulation PSK et FSK Turbo code (1,2,2)(7,5)8 LFSR pour Gold de 18 bits		Longueur du LFSR (bits) de 8 bits pour la démodulation PSK et FSK Turbo code (1,2,4)(23,33)8 LFSR pour Gold de 18 bits		Longueur du LFSR (bits) de 16 bits pour la démodulation PSK et FSK Turbo code de Polynômes générateurs du tableau IV.3 LFSR pour Gold de 18 bits	
Paramètres	Architecture proposée	Velcro	Architecture proposée	Velcro	Architecture proposée	Velcro
Number of Slice Registers (19200)	65	103	93	171	138	312
Number of Slice LUTs (19200)	47	53	59	79	87	134
Maximum Frequency (MHz)	644.300	Démodulateur de phase [90] : 1084.952 Démodulateur de fréquence [122]: 1025.010 Gold code: ------ Code convolutif 1025.010	644.300	Démodulateur de phase [90] : 911.826 Démodulateur de fréquence [122]: 1080.030 Gold code: ------- Code convolutif 1080.030	644.300	Démodulateur de phase [90] : 911.826 Démodulateur de fréquence [122]: 1080.030 Gold code: 644.330 Code convolutif 1080.030
Power (mW) Quiescent: Dynamic: Total :	379.12 2.38 381.50	379.10 1. 43 380.53	379.11 1.90 381.01	379.14 4.05 383.19	379.14 4.28 383.43	379.21 9.28 388.49

On remarque aussi que la fréquence maximale de l'architecture proposée est inférieure à une architecture matérielle dédiée (le tableau IV.4), mais le grand avantage est la flexibilité offerte qui augmente le parallélisme du traitement en partageant des ressources communes entre les algorithmes, ce qui conduit à améliorer la vitesse de calcul [1], [71].

La puissance consommée par l'architecture a été estimée par l'outil de Xilinx «XPower analyzer », après le processus « place and route ». Le tableau montre les rapports détaillés de la puissance, dans Xilinx Virtex5. La comparaison de notre architecture proposée avec l'architecture Velcro, indique que la notre réalise une consommation de puissance appropriée, notamment lorsque la taille des registres augmente (tableau IV.4).

- **Résultats de simulation**

Pour la vérification de nos implémentations, nous avons utilisé ISE.10 pour la simulation des résultats. Le but de cette simulation consiste à vérifier le fonctionnement global de l'architecture proposée. Un fichier testbench est écrit spécialement pour cette simulation. Les figures (IV.26) et (IV.27) montrent les résultats de simulation de l'implémentation proposée respectivement dans le mode démodulation et le mode codage.

Dans le mode démodulation, l'architecture peut mesurer la fréquence et la phase d'un signal modulé. Dans cette configuration, le signal « Mode » qui permet de configurer l'architecture, est égal à '1'. Sur le chronogramme de la figure (IV.26), le signal PSK à démoduler est noté « *Mn* » après conversion en binaire. Le signal FSK à démoduler est noté « *Si* » après conversion en binaire.

- Data_out_phase01 est la sortie du premier LFSR pour le démodulateur de phase.
- Data_out_phase02 est la sortie du deuxième LFSR pour le démodulateur de phase.
- Data_out_freque02 est la sortie du LFSR pour le démodulateur de fréquence.
- Data_out_freque est la valeur sortie du deuxième LFSR pour le démodulateur de fréquence.

Dans le cas de la démodulation de fréquence, on remarque sur le chronogramme que le registre LFSR$\oplus\gamma$ est incrémenté sur les fronts montants du signal (Si × *clk*). Lorsqu'un front descendant du signal *Si* est détecté, on lit la valeur d'état du registre LFSR$\oplus\gamma$. Dans cet exemple, cette valeur représente la sixième itération du registre LFSR$\oplus\gamma$ (de valeur 0800 en Hexa sur le chronogramme). La LUT de la figure (IV.14) est ainsi adressée avec cette valeur. En effet, le registre LFSR$\oplus\gamma$ fait six itérations, c'est-à-dire six périodes d'horloge (6 T_{clk}) pendant une demi-période du signal FSK ($\frac{T_{siganl}}{2}$ *ou* $\frac{1}{2f_q}$) après la

conversion binaire (voir annexe C). Par conséquent, nous avons $(6.T_{clk} \rightarrow \frac{T_{siganl}}{2})$, donc la fréquence mesurée résultante fournie par cette LUT est $f_q = \frac{1}{12} f_{clk}$.

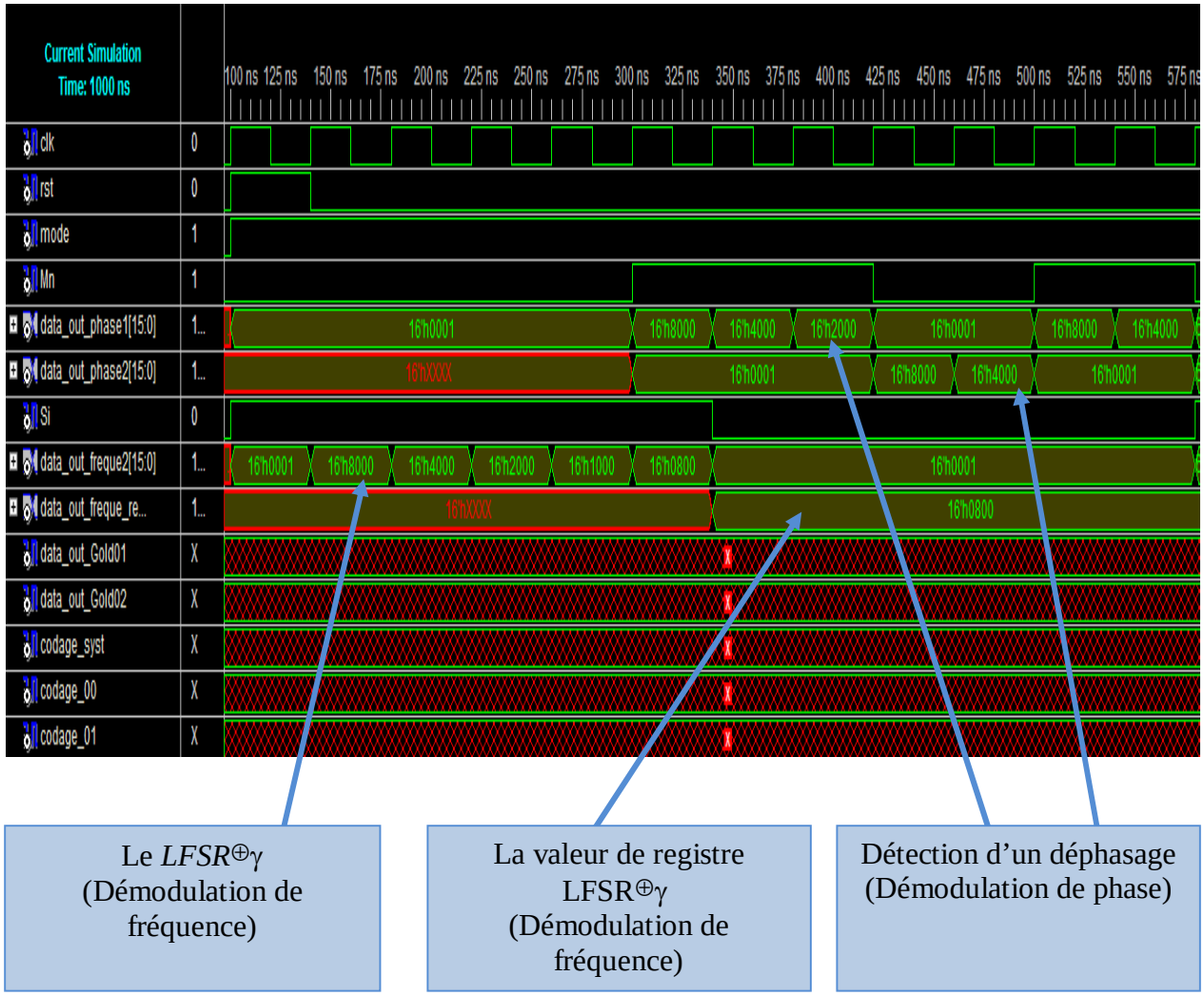


Figure IV.26. Résultats de simulation dans le mode démodulation.

Dans le cas de la démodulation de phase, on remarque sur le chronogramme qu'il y a un déphasage détecté par les deux registres, où le nombre d'itérations du registre LFSR⊕1 (qui permet de mesurer la durée pendant laquelle le signal $m(n)$ reste à l'état haut) et le nombre d'itérations du registre LFSR⊕2 (qui permet de mesurer la durée pendant laquelle le signal $m(n)$ reste à l'état bas) ne sont pas les mêmes. Le nombre d'itérations du registre LFSR⊕1 représente le nombre de cycles d'horloge pendant lesquelles le signal $m(n)$ reste à l'état haut, et le nombre d'itérations du registre LFSR⊕2 représente le nombre de cycles d'horloge pendant lesquelles le signal $m(n)$ reste à l'état bas. Donc, ceci permet de mesurer la valeur de la phase en utilisant l'équation (15) et la valeur de la fréquence mesurée par le registre LFSR⊕γ.